




# **TSC 6800 Debug Package**

**COPYRIGHT © 1978 BY  
Technical Systems Consultants, Inc.  
P.O. Box 2574  
West Lafayette, Indiana 47906  
All Rights Reserved**





### Copyright Notice

*This entire manual, source listing and documentation is provided for personal use and enjoyment by the purchaser. The entire contents have been copyrighted by Technical Systems Consultants, Inc., and reproduction by any means is prohibited. Use of this program, or any part thereof, for any purpose other than single end use is strictly prohibited.*



## Table of Contents

1. Debug Tutorial	1
I. Introduction	1
II. The Simulated Computer	1
III. Whats in Memory?	3
IV. Simulating the Program	6
V. Breakpointing the Program	7
VI. Advanced Breakpoints	9
VII. Protect Your Memory	12
VIII. Trapping Those Bugs	13
IX. And There is Still More!	14
2. Command Descriptions	17
I. Introduction	17
II. General System Control	17
III. Memory Commands	22
IV. Simulation Control	24
V. Breakpoints	28
VI. Memory Protection	32
VII. Execution Traps	33
VIII. Interrupt Control	35
3. Command Summary	37
4. Message Descriptions	39
5. Getting Debug Running	41
6. Example Use	43
I. Sample Program Source	43
II. Sample Debug Session	44
7. Adapting to Your System	47
I. I/O References	47
II. I/O Related Storage	47
III. Stack Pointer References	48
IV. The X Command	48
V. System Tables	48
VI. Saving the Altered Program	48
8. Relocating the Debug Package	49
9. Debug Package Source Listing	51

## Preface

The TSC Debug Package is a very powerful tool for assembler language program debugging. It offers the power and flexibility of an expensive hardware emulator at only a very small fraction of the cost! Used with care, this package will save many hours when debugging programs.

It is recommended that the entire user's manual be read before attempting any serious debugging. The 'Tutorial' is written to provide a fairly complete introduction to the Debug Package, while the 'Command Descriptions' is a very complete and concise description of all Debug features and commands. Consult 'Getting Debug Running' for details on how to get the program started. Working through the example given in 'Example Use' is a good place to start once the manual has been read.



## Debug Tutorial

### I. Introduction

Program debugging is usually thought of as work. It should be thought of as an art. There is no reason for a lot of crying while attempting to make a new program do what was intended. This is only true, however, if the program was designed with some forethought and planning. Computer programs are executed in a logical, step by step, fashion. This is the approach both program writing AND debugging should take. So many times a programmer will spend hundreds of hours, carefully planning the flow of a new program but spend only a few minutes thought on a debugging approach. The debugging is usually attempted in some hap hazard, keep your fingers crossed, method. Sometimes this works and sometimes it does not, but in most cases, valuable time is wasted.

By using a debugging tool and by incorporating some logical thinking, program debugging can become very straight forward and sometimes even fun! The purpose of this tutorial is to introduce the reader to the capabilities of the TSC Debug Package and offer some suggestions on how to tackle those program bugs. The following sections give a more detailed description of its capabilities.

### II. The Simulated Computer

The TSC Debug Package is more than the name may imply. It is in fact a complete 6800 simulator. A computer simulator is a program which when run, behaves exactly like the computer it is simulating. Given 6800 machine language, the simulator will perform the instructions exactly like the 6800 CPU. There are two major differences, one being an advantage, the second being a disadvantage. First for the good news. The simulator has the ability to keep close account of all internal actions. For example, any illegal opcodes are quickly detected and reported. Such things as stack overflow and underflow are also easily checked. Each byte of memory may have an assigned protection type such as write protection. General conditions may also be spotted such as the occurrence of a transfer of address type instruction. Overall, the simulator can keep close watch over the executing program and detect any peculiarities.

This all sounds great, but as stated before, there is a disadvantage in the simulator, namely speed. The simulated program runs somewhere between 100 and 300 times slower than a real 6800 CPU. This means that real time dependent code may not be simulated. This is not a serious drawback since less than one percent of all computer programs are real time dependent.



The 6800 simulator incorporated in the TSC Debug Package supports all of the 6800 instructions. All of the user registers are also provided (A, B, CC, X, PC, and SP). To examine the contents of these registers it is only necessary to type R followed by a carriage return. This is assuming the Debug Package is ready to work indicated by the two asterisk prompt ('\*\*'). Typing the R command will cause the debugger to display a line containing all register names followed by their contents in hex. At the end of the line is the instruction currently being pointed to by the program counter (P register) and it is displayed in disassembled form (standard Motorola mnemonics). A nonstandard register is also displayed, the N register. This register's value represents the subroutine nest depth. Each time a subroutine is called, its value will be incremented, and each time a return from subroutine is executed, its value will be decremented. The contents of any of the displayed registers may also be set by using the SET command. For example:

```
**SET,P=100,A=F3
```

will set the value of the PC to hex 100 and the value of the A register to hex F3. There are several other registers and states of the simulated machine. These can be viewed by typing MACH. The items displayed with this command are primarily the states of various traps which will be described a little later.

There are several other internal machine variables which may be easily examined. One of these is the contents of the stack. Typing STACK will display the top several bytes of the stack. If more stack contents are desired, simply type the number of items desired after the command.

```
**STACK,15
```

This will display the top 15 bytes of the stack. Note that a comma was used as a separator in the command line. It will be used in all examples in this manual but a space is also acceptable and sometimes easier to type. Another command which references the stack is the RET command. This will print the top two bytes of the stack as an address and represents the return address if currently in a subroutine.

The simulated machine always keeps track of where it has been and how much time was spent there. The machine 'states counter' is used to tally the total number of machine states or cycles used so far by the executing program. Each 6800 instruction requires a certain number of machine cycles to execute. If the CPU is running at 1 megahertz, each machine state is equivalent to 1 microsecond. The machine states counter is capable of counting up to 99,999,999 cycles, or roughly 99.99 seconds of actual program execution time. This counter is useful for determining the exact execution time of a routine.



The TRAIL command will print the address of the last transfer type instruction. A transfer of address instruction is one which causes the CPU to change its normal course of instruction execution. Normally instructions are executed in a sequential fashion, stepping through memory sequentially. A JMP instruction for example will cause the next instruction to be fetched from the address specified in the instruction, rather than from the next sequential address. In effect, we have a transfer of address. The TRAIL command will print the location of the last transfer type instruction that was executed. This is very handy in determining what caused a program to end up in memory where it did.

The simulated machine is capable of running in two different modes. These are referenced as mode 1 and mode 0. In mode 1 (the default mode), all checking and bookkeeping is performed. In mode 0, several of the features are turned off in order to improve the speed performance of the simulator. It is recommended that mode 1 always be used since it does the most work for you and will catch more errors.

### III. Whats in Memory?

Now that the simulated CPU has been described we need to look at memory. The TSC Debug Package offers several ways of examining the contents of memory locations, as well as altering them. The simplest form is the MEM command, or M for short. Typing M followed by an address will display that byte of memory. For example:

```
**M,100
100 CE
```

shows that memory location hex 100 contains a hex CE. At this time several choices are at hand. If all you wanted to do was check the contents of location 100, simply type a carriage return and the debug prompt will be issued. If you want to change the contents of 100, simply type the new value followed by a 'space'. The 'space' tells the debugger that the new value is ready to be entered. It is only necessary to type the significant digits of the new value to be entered. For example, if 6 was to be entered, simply type 6 followed by a space. It should be noted that only the last two digits will be used so if 'C23A' is typed, '3A' will get entered. If zero is to be entered, simply type a space. After the new value is entered, the next sequential memory location will be displayed. Any time a non hex character is typed (with the exception of space), one of two actions will occur. First if the character is a 'line feed', the previous location will be displayed, with the current location left unchanged. If the character is any other non hex character, the next location will be displayed leaving the current unchanged. An example will clarify the M command's use.



```
**M,100
0100 CE .
0101 3A 46
0102 4D
```

Location \$100 was left unaltered, while location \$101 was changed from a \$3A to \$46. Finally this mode was exited on the next line by typing a return.

Many times while program debugging it is desirable to examine a large block of memory. The DUMP command is used for exactly that. This command will display 16 lines of data, 16 bytes per line, for a specified memory region. Each byte is displayed as a hex value as well as its ASCII equivalent. All control characters (those bytes having a value less than 20 hex) are displayed as an underscore character '\_'. To display 256 bytes starting at memory location \$1000, the following command should be typed:

```
**DUMP,1000
```

At the end of the dumped block the program will stop and wait for a character to be typed. Typing an 'F' will move forward in memory, printing the next sequential 256 bytes. In this example, typing an F would display the block starting at \$1100. It is also possible to display the previous block of 256 bytes by typing a 'B', for backward movement. A carriage return will cause the debugger to regain control and the prompt will be reissued. Any other characters will be ignored. It should be noted that any time the debugger is displaying data on the terminal, the display may be stopped at the end of the line by typing an 'escape' character. Once stopped, another 'escape' will resume the display, while a 'return' will give control back to the debugger. This is a very convenient feature.

Another useful memory interrogation command is the FIND command which is used to find a specific string of bytes or characters in a selected block of memory. As an example, suppose there was a jump to subroutine instruction somewhere in your program. It is known that the code is BD 34 00, and that it is somewhere between locations \$100 and \$300. The following command line will find it.

```
**FIND,100,300,BD,34,00
```

This tells the debugger to look between memory locations hex 100 and 300 for the hex string 'BD3400'. All memory locations which contain this string will be displayed on the terminal. The length of the string searched is limited by the maximum command line length which is 80 characters. It is also possible to search for an ASCII string. Suppose it was necessary to find the character string 'ERROR 3' in memory. It should be somewhere between locations \$200 and \$1000. This can be done in the following way:



**\*\*FIND,200,1000,"ERROR 3**

The double quote character tells the find command that the following characters are to be considered ASCII characters instead of hex. Otherwise the command works exactly as described above.

So far the memory commands described have been oriented toward hex and ASCII values. Many times during debugging it is necessary to decode these hex values into assembler language instructions. The DIS command does exactly that! This command is a complete program disassembler which allows the user to examine the contents of memory in a higher level form. Each memory location in a specified block will be printed as address, followed by the opcode mnemonic and addressing mode. Standard Motorola mnemonics and addressing mode designators are used. To use the disassembler, simply type the command name (DIS), followed by two address boundaries. For example, to disassemble the memory range between locations 100 and 108, type the following.

```
**DIS,100,108  
0100 LDAA $32  
0102 STAA $0240  
0105 BNE $0121  
0107 DECA  
0108 STAA 2,X
```

Remember that at any time the display is being produced, the 'escape' key may be typed to temporarily halt the action. The DIS command is a very useful and powerful command.

Now that we can examine memory in a higher level form it would be nice if we could alter it in the same way, that is, using assembler language mnemonics. The ASM command does exactly that! It acts as a line at a time assembler, allowing standard mnemonics and addressing modes to be typed, while the corresponding hex values are automatically inserted into memory. To start this process simply type the command name followed by the address where the code should be placed. The debugger will respond by printing the address of the location specified followed by a space. At this time, simply type the desired instructions following each with a carriage return. The next available address will then be printed and assembly can continue. Typing a carriage return in response to the address prompt will exit this mode of operation. To show the workings of this command, some code will be assembled at location \$200.

```
**ASM,200  
0200 LDAA 10  
0202 LDAB $10  
0204 PSHA  
0205 LDAA 'M  
0207 STAA 0,X
```



```
0209 JMP $3000
020C
**
```

Note that numeric values are interpreted as decimal unless preceded by a dollar sign (\$) to designate hex. It is also possible to enter an ASCII constant by preceding it with a single quote ('). No spaces are allowed between the register specifiers 'A' or 'B' and the instruction (e.g. LDAA is correct, LDA A is not). The ASM command is a great time saver!

#### IV. Simulating the Program

Program simulation is very simple. If the test program starts at \$100, simply type START,100 to start the simulation process. The program will run exactly as the CPU would run it, just slower. The START command clears several of the machine conditions such as the states counter. To start a program where it left off, the GO command can be used. This will cause the program to start execution at the location pointed to by the program counter (P register). No states will be cleared.

A very valuable feature of the simulator is the 'trace mode'. When trace is enabled, a register dump (exactly like that produced by the R command) will be displayed after each instruction is executed. The simulation may be temporarily halted by typing an 'escape' character anytime during the tracing operation. The simulation may also be stopped by typing a 'control C'. This will cause the debug prompt to be reissued. To enable the trace mode use the TRACE command.

```
**TRACE=10
```

This line will cause the debugger to trace all instructions which are in a subroutine nest level of 9 or lower. The number in the command line specifies the nest level where tracing should be disabled. This allows only the outermost program structure to be traced if desired, while the deeper subroutines will be simulated without the tracing. To disable the trace, use a count of zero (e.g. TRACE=0).

There are several other methods of starting program simulation. One is the SIM command. This command will allow the simulation of a specified number of instructions. Tracing is disabled during the execution of this command.

```
**SIM,100
```

This line will cause 100 instructions to be simulated starting at the address pointed to by the program counter. The TSIM command is identical to the SIM command except trace is automatically set to 256 during the execution of the command.



It is often desirable to step through the execution of a program, one instruction at a time. The STEP command will start simulation at the instruction pointed to by the program counter, execute a specified number of instructions, print a register dump, and then wait for input. At this time, a space will repeat the process, while a return will return control back to the debugger. The usual method of operation is 'single' step which will execute one instruction, then dump the registers. This mode can be entered by:

**\*\*STEP**

Multiple instructions can be executed between register dumps by specifying a count. For example;

**\*\*STEP,25**

will cause 25 instructions to be simulated at a time. The step mode is a very powerful method for closely following the flow of a program.

During program execution, the simulator keeps track of the last 256 instructions executed. If a program ever goes off on its own, ending up in memory where it should not, the PAST command can be used to examine the instructions executed to get it there. Typing the command,

**\*\*PAST,20**

will display the addresses and mnemonic instructions of the last 20 opcodes executed. This feature alone will find a good percentage of program bugs.

## V. Breakpointing the Program

So far, methods have been described which allow all or a certain number of instructions to be simulated. Most of the time, the number of instructions to a certain point in the program is not known. It would be helpful if a break in the program simulation could be specified to take place at a particular point in the program, or in other words, breakpoints. A breakpoint is a mechanism for stopping the execution at a specified address in the program. As an example, to set a breakpoint at location \$23A, use the following command.

**\*\*B@23A**

As the program executes, any time location \$23A is reached, simulation will stop and the registers will be dumped to the terminal. After the program has stopped, typing a 'G' will restart execution, starting at address \$23A (the breakpoint will be temporarily ignored). It should be noted that the method used to create the breakpoint does not alter the contents of memory in



any way. This means that after setting a breakpoint, the contents of memory at the breakpoint location will be unchanged. This allows breakpoints to be set in ROM as well as RAM!

In the above example, the breakpoint caused two actions to take place. One was printing the registers, the other was stopping program simulation. These actions are the ones performed by most debugging systems. The TSC Debug Package allows six other actions to be performed upon the execution of a breakpoint. A list of all 8 possible actions follow:

1. R...Print register contents
2. Z...Zero the states counter
3. T...Enable the trace function
4. U...Disable trace (untrace)
5. H...Histogram counter
6. M...Print a message
7. J...Jump to specified address
8. S...Stop simulation

The first breakpoint example shown defaulted to R and S type actions since none were specified. The Z action zeroes the machine states counter. This is useful for program timing. For an example, the states counter may be zeroed upon entry to a subroutine and a stop type breakpoint set at the exit point of the routine. By using the STATES command after the program stops, the exact number of executed machine states for that routine will be displayed.

The T and U actions allow the trace mode to be enabled and disabled at selected points in a program. When enabled, trace will be set to level 255. Many times, tracing is only desired during one routine or selected portion of the program. These actions will permit this sort of program tracing. A few examples will demonstrate action type breakpoints.

```
**B,RZ@1000
**B,T@A16
```

The first command will set a breakpoint at location hex 1000 which when executed will print the registers and zero the states counter. The program will then continue since a stop (S) action was not specified. The second example will cause trace to be turned on at location hex A16.

Another action is the histogram (H). A histogram counter counts the number of times the instruction at that address has been executed. This is useful for determining 'hot spots' or sections of programs which are executed very frequently. By setting a histogram breakpoint at the first instruction of each subroutine in a program, it is possible to find out exactly how many times each routine was called. As an example, suppose there were three subroutines in a program, and they were located at \$100, \$123, and \$1A0. To set histogram counters at these



locations, type the following commands:

```
**B,H@100
**B,H@123
**B,H@1A0
```

After simulating the program, typing HIST will display the totals of the counters at each address. This command is used to examine the histogram counters at any time. The CLH command is used to clear the histogram counters.

```
**CLH,100
**CLH
```

The first command will clear (set to zero) the value of the histogram counter at location 100. The second command will zero all of the counters. The histogram commands allow a very complete profiling of a program, letting the user 'fine tune' it for maximum speed.

The remaining two action codes are special purpose. One permits a selected message to be printed as the action, the second allows transfer of control to a specified address (like a JMP instruction).

```
**B,M@325,SUB 1
**B,J@27C,1000
```

The first line will print the message "SUB 1" each time the instruction at \$325 is executed. The second command will cause the instruction at address hex 1000 to be the next instruction executed. The instruction at 27C will not be executed!

Any combination of action codes may be listed for a breakpoint. They are executed in the order they appear in the above list. For example,

```
**B,TRZ@300
```

will cause the registers to be displayed (R), the states counter to be zeroed (Z), and trace to be enabled (T), in that order. This ordering may be important, for in the actions 'RSJ', the stop (S) will never get executed since the J transfers control to another address.

## VI. Advanced Breakpoints

Programs containing loops or recursion are often difficult to breakpoint since one particular section of code may be called thousands, or even millions of times. As an example, suppose there is a loop in the program being debugged, and it is necessary to examine the contents of the X register after the 600th time through the loop. One way is to set a breakpoint at



the desired instruction and start the program simulating. Every time the program halts at the breakpoint, type G to restart it. Repeat this process 600 times and you can examine X. You are probably thinking that this would take forever and you are right! The TSC Debug Package allows a pass counter to be associated with a breakpoint. This count determines how many times the instruction at the address of the breakpoint should be executed before the actions specified should be performed. In the above example, assuming the instruction to be breakpointed is at address 300, the following will do exactly what we want.

```
**B@300,>600
    or
**B,SR@300,>600
```

Both commands are identical since the first defaults to SR actions. The '>' is the pass count modifier and should be read as 'after'. The result of this command is to stop and print the registers on the instruction at location 300, after 600 times through it. Once the count reaches 600 (or whatever value was set), the breakpoint actions will always occur. A second similar type of pass count uses a '<' for a modifier and should be read as 'before'. This is used to create a temporary breakpoint.

```
**B,R@300,<100
```

This command will set up a breakpoint at 300 which will print the registers for the first 100 times through. After the 100th time, the breakpoint will be cleared and no longer function. In summary, the pass count value associated with a breakpoint is decremented each time the instruction at the specified address is executed. If the modifier is a '>', no actions will be performed until 'after' the count has reached zero. With the '<' modifier, actions are only performed 'before' the count reaches zero, and once it is zero, the breakpoint is cleared.

In the above example it was decided that the program should be stopped after 600 times through the loop. While debugging loops, it is not always possible to determine an exact number of times to execute the loop before it should be stopped. Often it is desirable to stop on a certain condition, such as the contents of a register or the state of a particular memory location. Conditional expressions are allowed in breakpoint definitions and yield a great deal of power. The conditional can be determined on the contents of a selected register (A, B, C, X, P, S, or N) being equal (or not equal) to a specified value. A particular memory location may also be tested for zero or not zero. Following are a few examples.

```
**B@1000,IF A=3F
**B,R@320,IF B!=10
**B,T@6A7,IF $20=0
```

The 'IF' statement designates the conditional part of the



breakpoint definition. The first example will stop and print the registers at location hex 1000 but only when the value in the A accumulator is hex 3F. The second example will print the registers at 320 only if the contents of the B register is not hex 10 ('!=' is to be read as 'not equals'). The last example will enable the trace mode at location 6A7 if the contents of memory location hex 20 is zero. The dollar sign '\$' is used to designate a memory reference and not a hex value (the value is always interpreted as hex). The value on the right of the equals sign must always be zero when a memory reference has been designated.

The above breakpoint features may be combined in a variety of ways to produce an almost endless variety of breakpoints. As an example:

```
**B,TZ@1000,>100,IF X=100
```

will cause trace to be enabled and the states counter to be zeroed, after executing the instruction at hex 1000, 100 times, but then only if the value of the index register is \$100. It should be noted that the H, M, and J action codes will not allow a conditional expression as part of the breakpoint definition, and J will not support a pass counter.

Once breakpoints are set it is possible to examine the location of them as well as remove them. To check the locations of breakpoints, use the BP command.

```
**BP
**BP,100
**BP,100-500
```

The first line will print the location of all breakpoints, each one followed by a list of its action codes. No pass counts or conditionals are displayed. The second example will display the action codes of the breakpoint at location hex 100 (if one exists). The last command line will display all breakpoints between location 100 and 500, inclusive. The CLB command is similar in syntax but is used to clear or remove a breakpoint. CLB by itself will clear all breakpoints. If it is followed by an address, the breakpoint at that address will be removed. If two addresses are specified, then all breakpoints in their range will be cleared.

While debugging very large programs, it may become quite time consuming to simulate the program up to a desired address. For example, a program which requires a minute to execute in real time may require over an hour if simulated. To get around this problem, it is possible to set a 'real time' breakpoint. This is entirely different from the previously described breakpoints in that it does modify the contents of memory (by substituting a JMP instruction) and no pass counting or conditionals are permitted. The only action performed is to stop and print the registers. An



example of use follows:

```
**RT,5A00
```

This command will cause the CPU to start executing the program (NOT the simulator) at the current address of the program counter. When the program reaches the specified address (5A00), the program will stop, print the registers, and restore the contents of RAM at that location (remove the breakpoint). Since the program is being executed in real time and not being simulated, no other breakpoints, illegal condition checking, states counting, or record keeping is performed. This type of execution is not recommended for this reason and should only be used where the simulation time gets tremendously long.

## VII. Protect Your Memory

Perhaps the most aggravating aspect of program debugging is having your program destroy itself in memory. Too many times, programs 'run away', writing garbage in memory, usually exactly where it is not wanted. In these instances, it would be nice to be able to 'write protect' memory, or at least certain portions of it. The TSC Debug Package will allow exactly that! In fact, any section of memory, right down to a single byte, may be write, execute, memory, or simulate protected! Write protecting memory will prohibit any stores or writes into it. Execute protection prohibits opcodes from being fetched from memory. In other words, the program counter (PC) will not be permitted to point to a location of memory which is execute protected. Memory protect is a brute force type of protection. By memory protecting a region, you are in effect saying that no memory exists in this region and that nothing should be allowed to reference it in any way. Any memory referenced in conflict with its protection will cause the simulation to stop and an appropriate message will be printed. Finally, simulate protection is slightly different from the rest. It is used to tell the simulator to execute any code in a simulate protected region in real time, or in other words, not simulated. A restriction requires the code in a simulate protected region to be called as a subroutine (JSR or BSR) from the non-simulate protected code. This is very convenient for I/O operations. All I/O routines can be simulate protected (such as TTY and disk routines) allowing them to be executed by the CPU (real time) and not the simulator. It is often convenient to simulate protect the entire region of memory containing the monitor and/or operating system since this code is known functional. Keep in mind that code in simulate protected memory may only be accessed via a subroutine call.

The command used to set protection is PROT. A few examples will demonstrate its use.



```

**PROT,100-3FF,X
**PROT,2E0,W
**PROT,500-6FF,M,1200-1FFF,W

```

The first example will execute (X) protect the memory between locations \$100 and \$3FF. The second line write protects (W) location \$2E0. The last example will memory protect (M) locations \$500 through \$6FF and write protect \$1200 through \$1FFF. There are some general rules to follow when protecting memory. Memory protection should be used on all sections of memory not referenced or used by the program being debugged, especially the area of memory containing the Debug Package. This will keep a runaway program from clobbering something it should not. Sections of memory which are used for register storage or flags should be execute protected. Memory containing the actual program code should be write protected for obvious reasons. Finally, as mentioned above, the memory locations where the monitor and/or operating system reside should be simulate protected.

Once the protection has been defined it may be checked by using the BOUNDS command. This command will allow the examination of the boundaries of each type of protection. Either all types or selected ones may be displayed.

```

**BOUNDS
**BOUNDS,W,M

```

The first example will display all types while the second will show only the defined boundaries for write and memory protection. Memory protection can be cleared in a similar fashion.

```

**CLP
**CLP,X,W

```

The first command will clear all protection while the second will only clear the defined execute and write protected regions.

#### VIII. Trapping Those Bugs

The previously described breakpointing feature allows programs to be stopped at specific locations and on specific conditions. It is often desirable to 'trap' a program on some general condition such as every time a transfer of address instruction is encountered. The memory protection described above is a form of trap in that the program will stop if a protection violation is detected (e.g. writing into write protected memory). There is address information associated with this protection which makes it different from the general traps available in the Debug Package. The general traps cause programs to stop on a general condition which is not address dependent.



One of these traps is the illegal opcode trap which is always enabled. Any time an illegal opcode is encountered during the course of program simulation, the program will stop and report its occurrence. A second, always enabled trap will stop the program if an RTS instruction is encountered and the current nest level is 0.

There are several user controlled traps which may be enabled and disabled at will. The transfer trap is enabled with the XFR command. When enabled, the program will stop each time a transfer of address is encountered. These instructions are JMP, BRA, and all conditional branches such as BCC. The subroutine calls and returns are not trapped out.

```
**XFR=ON
```

```
**XFR=OFF
```

These two commands will enable and disable this trap respectively. Once a program has stopped because of a transfer trap, typing G will restart it, allowing the current transfer to be executed. This is very useful for quickly following the major flow of a program. Another one of the general traps allows halting the program if the subroutine nest counts reaches a specified level.

```
**NEST=20
```

This will cause a trap if the nest level ever reaches 20. To disable the nest trap, use NEST=0.

The last general trap to be discussed is the ITRAP. This command allows activation of the interrupt trap and will cause the simulating program to stop if an interrupt type instruction is encountered (SWI, RTI, and WAI). Since these instructions are not used in the majority of programs it is a good idea to use this feature. An example will demonstrate its use.

```
**ITRAP=ON
```

```
**ITRAP=OFF
```

These two commands will enable and disable the interrupt trap respectively.

#### IX. And There is Still More!

There are still many undescribed features of the TSC Debug Package. One of these is the handy little CALC command which acts as a hex calculator. Typing CALC followed by a return will cause the debugger to output an equals sign (=) for a prompt. At this time hex and decimal addition and subtraction may be performed. To add two numbers simply type them in separated by a plus sign. If the number is hex precede it with a dollar sign, otherwise the debugger will interpret it as decimal. Use a minus sign for



subtraction. It is also possible to do base conversions. This can be accomplished by entering just one number after the prompt (hex or decimal) followed by a return. All answers are displayed in both hex and decimal. An example follows.

```
**CALC
=$1A+10
$0024 36
=256
$0100 256
=
```

After entering the calculator mode, the numbers hex 1A and decimal 10 were added to give the result hex 24 or decimal 36. The second entry is a base conversion of the decimal number 256. The result shows its hex equivalent is \$100. The calculator mode can be left by typing a return in response to the prompt.

There are still many other features in the Debug Package, such as interrupt simulation, which have not been described. It is not the intention of this tutorial to teach all there is to know about the debugger, but to teach enough to make the user feel comfortable with the majority of its features. Once the material in this section is thoroughly understood, the following detailed command description should be studied in depth.

Now that the basic mechanics of the Debug Package are understood they should be put to good use. Keep in mind that a logical and planned approach should be taken when debugging a program. Use the available tools such as memory protection and breakpoints. When first starting the debug process on a new program, start at the beginning, working your way through the flow of the program. Let the program be the guide. If you pay close attention, it will definitely point out the bugs. Above all, have patience. Great bugs are not killed overnight!





...the ... of the ...  
 ...the ... of the ...  
 ...the ... of the ...

100-100000  
 100-100000  
 100-100000  
 100-100000  
 100-100000  
 100-100000

With respect to the ...  
 ...the ... of the ...  
 ...the ... of the ...

There are ...  
 ...the ... of the ...  
 ...the ... of the ...

Now that the ...  
 ...the ... of the ...  
 ...the ... of the ...

100-100000  
 100-100000  
 100-100000  
 100-100000  
 100-100000  
 100-100000





## Command Descriptions

## I. Introduction

This section of the manual contains a detailed description of each Debug command. Each command is shown with a few examples. The syntax definitions show optional items in square brackets ([ ]). All command parameters are shown separated by commas for clarity in the syntax definitions and examples. Any place a comma is shown, a space may also be used. The following definitions apply throughout this document:

<u>Item</u>	<u>Meaning</u>
<address>	1-4 digit hex value
<value>	decimal number (max = 255)
<count>	decimal number (max = 65,000)

The Debug Package is ready to accept a command anytime the '\*\*\*' prompt is present on the line. When typing commands, a 'control H' will cause a backspace, and delete the last character typed. A 'control X' will cause the entire line to be deleted and a new prompt of '???' will be output to show the deletion of the line. Any time text is being output to the terminal, display may be stopped at the end of a line by typing an 'escape' character. Once stopped, another 'escape' will restart the output while a 'return' will give control back to the debugger and the '\*\*\*' prompt will be output.

## II. General System Control

The general system control commands allow a variety of general actions to be performed. Register examination and changing is supported by use of the REG and SET commands. The status of several machine control registers can be obtained through the MACH command. Commands to view the stack contents, set simulation speed, reset machine parameters, enter a calculator mode, examine the 'machine states counter', and exit the debugger are all described in this section.

## C[ALC]

## PURPOSE:

The calculator mode will be entered and a '=' prompt will be printed. The calculator will allow addition or subtraction of two numbers. The numbers may be hex (designated by a '\$' prefix) or decimal. If two numbers are typed, they must be separated by a '-' or '+' and the appropriate result will be displayed. The answer is shown in both hex and decimal. It is possible to enter



only one number (hex or decimal) followed by a return. The answer will be this number printed in both hex and decimal, thus allowing base conversions. After each calculation, a new '=' prompt will be output. To exit this mode, type a 'return' as a response to the prompt.

**EXAMPLES:**

CALC  
=\$A+10  
\$14 20

Enter calculator mode  
Add hex A and 10  
The result is printed

DEL[AY]=<value>

**PURPOSE:**

This will set the simulation delay (the amount of delay after each instruction is executed) to an amount proportional to <value>. The higher the number (max = 255) the longer the delay. A delay of zero will result in the delay being turned off.

**EXAMPLES:**

DELAY=100  
DELAY=0

Set delay to 100  
Disable the delay

DEP[TH]

**PURPOSE:**

The depth command will print the deepest value of the stack pointer (the lowest memory address at which the stack was extended during program simulation). To initialize this pointer, it is necessary to set the stack pointer using the SET command. The depth value will be set to the same value as the stack pointer. This command is useful for determining the amount of stack space required by a program.

**EXAMPLES:**

DEPTH

Print the deepest stack location

E[EXIT]

**PURPOSE:**

Exit the debug program. Use this command when finished with the Debug Package.

**EXAMPLES:**

EXIT

Exit the debugger



FL[AG][=&lt;address&gt;]

## PURPOSE:

The Flag register is a 2 byte word at the specified memory location which will be displayed on a REG command or during tracing, as the 'F' register. The memory location for the flag will be set to the address specified. If no address is given, the flag register will be disabled. This is useful for tracking flags in memory during program tracing. See the REG command.

## EXAMPLES:

FLAG=1A85      Set flag register to \$1A85  
 FLAG          Disable flag register printout

IND=ON or OFF

## PURPOSE:

Used to enable or disable the indirection printout in a register dump (see REG). If IND is ON, the register dump will show a register called 'I' which is the value of the memory location pointed to by the index (X) register. If this feature is off, the I register will not be displayed.

## EXAMPLES:

IND=ON      Turn indirection on  
 IND=OFF     Turn it off

MA[CH]

## PURPOSE:

The MACH command will print the current status of the simulated machine. Values displayed are for mode (M), trace (T), instruction count trap (I), nest trap (N), stop address (S), interrupt trap (IT), transfer trap (XT), IRQ count (IRQ), and NMI count (NMI). The description of these appear elsewhere in this manual.

## EXAMPLES:

MACH      Print the machine status

MO[DE]=1 or 0

## PURPOSE:

The debugger has two modes of operation, mode 0 and mode 1. The system comes up in mode 1. Mode 1 offers all debug features allowing the simulated program to run approximately 250 times slower than real time. In mode 0, the program will run approximately 100 times slower than real time, but the following features are not supported; nest count checking, all traps, states counting, memory protection, past instruction bookkeeping, and automatic interrupts. Mode 1 should be



used most of the time to take full advantage of the debugger.

## EXAMPLES:

MODE=1            Set mode to 1  
MO=0             Set mode to 0

## R[IEG]

## PURPOSE:

Print the contents of the machine registers. All values are shown in hex. Besides the condition codes (C), A, B, and X registers, program counter (P) and stack pointer (S), the nest level, N, is displayed (shows how deep in subroutine calls) as well as two optional registers. One is enabled by the IND command and displays the byte of memory being pointed to by the index register. This is shown as 'I' in the REG dump. The second option is enabled by the FLAG command and will display the selected two bytes of memory. This is shown as 'F' in the dump.

## EXAMPLES:

REG             Display all registers  
R               Display all registers also

## RES[ET]

## PURPOSE:

The RESET command is used to reset all machine states. All registers will be set to zero, the stack pointer will be set to \$A07F, all breakpoints and memory protection will be cleared, and the mode will be set to 1. This will set up the machine exactly the same as initializing the debugger upon first entry.

## EXAMPLES:

RESET           Reset the machine

## RET

## PURPOSE:

Print the top two items on the stack. If the system is currently in a subroutine, these bytes will represent the return address from this routine. If the nest level is currently zero (N=0), the message "NEST LEVEL IS 0" will be displayed.

## EXAMPLES:

RET             Print the return address



## S[ET],&lt;register list&gt;

## PURPOSE:

The SET command is used to set or assign values to registers. The <register list> is a list of register names (C,A,B,X,S,P,N) followed by an equals sign, followed by the hex value. Setting the stack pointer will also set the depth value to the same amount.

## EXAMPLES:

SET,P=100,A=C3  
S B=20 X=1FFF

Set PC to \$100 and A to \$C3  
Set B to \$20 and X to \$1FFF

## STACK[,&lt;value&gt;]

## PURPOSE:

Print the contents of the stack. The number of bytes specified by <value> will be printed. If <value> is not specified, the top 6 bytes will be printed. The stack is printed from high address to low address, so the top of stack will be the last item printed.

## EXAMPLES:

STACK  
STACK, 10

Print the top 5 stack bytes  
Print the top 10 stack bytes

## STAT[ES]

## PURPOSE:

Display the current value of the states counter. This value represents the number of actual machine cycles (micro seconds on a 1 megahertz computer) which have been executed since the last START or RESET command. It is also possible to set this counter to zero using breakpoints.

## EXAMPLES:

STATES

Print the current states count

## TRAIL

## PURPOSE:

Print the address of the last executed instruction which caused a transfer of address (e.g. JMP instruction). This is useful when attempting to find how a program ended up where it did.

## EXAMPLES:

TRAIL

Print the last transfer address



X,<operating system command>

PURPOSE:

The X command is only operational on disk systems (see Adaption). It allows the execution of any DOS command from the debugger.

EXAMPLES:

X,CAT,1                      Catalog drive 1

### III. Memory Commands

The memory commands allow examining and altering the contents of memory in a variety of ways. The assembler allows simple, direct insertion of object code by using standard opcode mnemonics and addressing mode designators. The disassembler provides an opposite type of convenience, in that the contents of memory may be displayed as assembler language mnemonics and operands. A single byte memory examine and change function is also available (the MEM command). Commands for viewing large blocks of memory, finding specific hex or ASCII strings, and filling a section of memory with a selected character are all available in this group.

A[SM][,<address>]

PURPOSE:

Enter the line at a time assembly mode. Assembly will start at the address specified or at the location of the program counter if no address is specified. No labels are permitted. All standard Motorola opcode mnemonics are accepted (no pseudo ops). When instructions contain a register specifier, there should be no space between the mnemonic and the specifier (e.g. LDAB, not LDA B). All standard addressing modes are accepted. All page zero references will be assembled as extended addresses. Three types of constants are permitted, decimal, hex (precede the number with '\$'), and ASCII (precede the ASCII letter with a single quote (')). The PC is automatically advanced to the next location after the line is assembled. To exit this mode, type a return in response to the address prompt.

EXAMPLES:

ASM,100	Start assembly at \$100
100 LDAA #10	Load A with 10
102 LDAB #'1	Load B with ASCII 1
104 BRA \$100	Loop forever
106	Exit with return



DIS],<start address>,<stop address>

PURPOSE:

Disassemble memory between the addresses specified. The address, mnemonic, and addressing mode will be printed out for each instruction in the range. If an illegal opcode is found, three stars (\*\*\*) will be displayed instead of a mnemonic, followed by the hex value found at that address.

EXAMPLES:

DIS,100,1A0            Disassemble from 100 to 1A0

DUMP],<address>

PURPOSE:

Dump 256 byte blocks of memory starting at the address specified. The memory is displayed 16 bytes per line, followed by the ASCII values of the hex numbers. After each block is dumped, typing an 'F' will move Forward and display the next 256 bytes, typing a 'B' will move Back and display the previous 256 bytes. Typing a 'return' will exit this mode.

EXAMPLES:

DUMP,A00            Dump memory at \$A00

FILL],<start address>,<stop address>[,<byte>]

PURPOSE:

This command will fill memory with the <byte> (hex) specified starting at the first address, filling through the second address. If <byte> is not specified, zero will be used.

EXAMPLES:

FILL,100,300,FF      Fill with FF from 100 to 300  
FILL,0,100            Clear from 0 to 100

FIND],<start address>,<stop address>,<string>

PURPOSE:

Find the specified string in memory. The search will start at the <start address> and continue through the <stop address>. The address of each location where the string is found will be displayed. The <string> can be entered in one of two ways. The first can be a string of hex digits separated by spaces or commas. The second is an ASCII string preceded by a double quote character. The limit on string length is the input buffer (72 characters).



## EXAMPLES:

FIND,0,60,7E,33,A2 Find the hex value 7E33A2  
 FIND,0,1000,"TEST Find TEST in memory

M[EM],<address>

## PURPOSE:

Examine and alter memory. The address specifies the first location to be examined. Upon entering this command, the address specified and its contents will be displayed on a new line. At this time, typing any non hex printing character will move to the next location and display its contents. Typing a 'line feed' will move to the previous location. A carriage return will exit this mode. To change the contents of a location, type the new hex value immediately following the one displayed. After the value, type a space. The new value will be entered and the next memory location will be displayed. It should be noted that it is only necessary to type the number of significant digits and only the last two digits are used. For example, typing a 1 would enter 01, typing 1A2 would enter A2, etc. If only a space is typed (no number) a zero will be entered. Any time a non-hex character is typed (besides a space), the next location will be displayed, leaving the current location unchanged.

## EXAMPLES:

MEM,540  
 M,200

Examine memory at \$540  
 Examine location \$200

## IV. Simulation Control

This group of commands is used to control the program simulator. Code in RAM or ROM may be simulated. There are several methods of initiating simulation. Programs may be executed with 'trace' on or off. While trace is on, each instruction will be displayed prior to its execution, along with the current state of the CPU (all register contents are displayed). Trace provides a very powerful tool for following program flow. Several keyboard commands may be invoked during actual program simulation. These commands allow the speeding up or slowing down of simulation, as well as ways to halt the execution of the program. The PAST command is a powerful bookkeeper which keeps track of where your program has been.



G[O]

## PURPOSE:

Start the program executing at the location currently pointed to by the program counter. No machine values are altered with this command.

## EXAMPLES:

GO  
G

Start the simulation at the PC  
Does the same thing

J[UMP],&lt;address&gt;

## PURPOSE:

This command is exactly like GO except execution will begin at the address specified. No machine values are altered with this command, except the program counter which is set to <address>.

## EXAMPLES:

JUMP,322  
J,80

Start simulation at \$322  
Start simulation at \$80

PA[ST][,&lt;value&gt;]

## PURPOSE:

Display the past several instructions executed by the simulated program. If <value> is not specified, the past 255 instructions will be printed (oldest to most recent), otherwise <value> sets the number of instructions to be displayed. Each instruction is shown in a disassembled form, with its address.

## EXAMPLES:

PAST  
PAST,10

Display the past 255 instructions  
Display the past 10 instructions

SIM[,&lt;count&gt;]

## PURPOSE:

Simulate the number of instructions specified by <count> with the trace disabled. If the count is not specified, one instruction will be executed. Execution starts at the current PC. No machine values are altered prior to simulation. Trace will be reset to its original value following SIM's termination.

## EXAMPLES:

SIM  
SIM,100

Simulate one instruction  
Simulate 100 instructions



ST[ART],&lt;address&gt;

## PURPOSE:

Start program simulation at the specified address. The PC will be set to the address specified, the states counter will be zeroed, and the nest count will be cleared.

## EXAMPLES:

START,1000

Start simulation at \$1000

ST,2A

Start simulation at \$002A

STEP[,&lt;count&gt;]

## PURPOSE:

This command will cause the debugger to enter the 'step' mode. The <count> specifies how many instructions should be executed at a time in this mode and defaults to one (single step). Upon entering the STEP command, the system will immediately execute the number of instructions specified by <count>, then print a register dump. The execution will begin at the location pointed to by the P register (program counter). After the register dump, typing a 'space' will cause execution of the next <count> instructions and produce another register dump. Typing a 'return' will exit the step mode. Any other character will be ignored. It should be noted that while in the step mode, breakpoints and tracing are inoperable.

## EXAMPLES:

STEP

Enter 'single step' mode

STEP,10

Execute 10 instructions at a time.

T[RACE]=&lt;value&gt;

## PURPOSE:

Set the trace depth. If value is set to zero, trace mode will be disabled. Setting trace to a non-zero value will enable tracing up to but not including the subroutine nest level indicated by <value>. For example, if TRACE=2 is entered, tracing will occur at nest level 0 and 1 but will be disabled at nest levels of 2 and higher. The nest level is displayed as 'N' in a REGISTER dump.

## EXAMPLES:

TRACE=255

Enable trace at all levels

T=0

Disable trace mode



TS[IM][,<count>]

PURPOSE:

This command is similar to SIM except trace mode is enabled (Trace=255) and the registers will be dumped after each instruction simulated. The count will default to 1 if not specified. Trace will be reset to its original value following TSIM's termination.

EXAMPLES:

TSIM	Trace and simulate 1 instruction
TSIM,20	Trace 20 instructions

'Control C'

PURPOSE:

Anytime a program is being simulated, a 'control C' will cause the execution to halt and the message 'OP HALT AT XXXX' to be displayed at the terminal. This means 'Operator Halt' and the XXXX will be replaced by the actual address where the program was halted.

'Escape Character'

PURPOSE:

During program tracing, typing an 'escape' will cause the program to pause at the end of the next displayed line. At this time, typing another 'escape' will enable the trace to restart, while typing a 'return' will return control back to the command entry mode.

'Control F'

PURPOSE:

During program simulation, the delay value (see DELAY) may be dynamically changed. Each time a 'control F' is typed (only during program simulation) the delay value will be decremented by one, thus making the program run faster. If the delay is zero, the 'control F' will be ignored. It should be noted that for large delays, many 'control F' functions will need to be typed to see the increase in speed.

'Control S'

PURPOSE:

This is similar to the 'control F' key but makes the simulation run slower. If the delay is already at its maximum value, the 'control S' will be ignored.



## V. Breakpoints

Breakpoints allow the insertion of check points into a program. A breakpoint always has an address associated with it. The address specifies where in the program the breakpoint action should occur. These actions range from printing the machine registers to controlling trace mode. Each breakpoint may also have a pass counter which determines the amount of time until it becomes active, or the amount of time it should remain active. The actions are also dependent on the result of a conditional expression involving a CPU register or memory location. Breakpoints are decoded with the following precedence. If the address of the current PC matches the address of a breakpoint, then the pass count is checked. If the counter is in a state to allow continuing, then the condition is checked (if present). Finally the actions specified for the breakpoint are performed. The other commands in this group allow clearing breakpoints (removing them), printing histogram counter values, print breakpoint location and type, and clear histogram counters.

B,<actions>@<address>[,<modifier><count>][,IF<condition>]

or

B@<address>[,<modifier><count>][,IF<condition>]

## PURPOSE:

The B command is used to set breakpoints. These breakpoints are nondestructive in that they do not alter the contents of memory at the breakpoint location. Two forms of the command exist. The first is the general form of the command and allows user definable breakpoint actions. The <actions> may be any one or combination of the following:

- R...Print register contents
- Z...Zero the states counter
- T...Trace mode on
- U...Trace mode off (untrace)
- H...Histogram counter
- M...Print message
- J...Jump to new address
- S...Stop simulation

The above actions are executed in the order shown. A histogram action causes a counter to be set up such that each time the instruction at the address specified is executed, the counter will be incremented by one. By later requesting a HISTogram, all of the counters and their associated counts will be displayed. The second form of the B command is a special case of the first. In this form, no actions are specified, and they default to S and R (just as if S and R were used in form one). The <count> part of the syntax is optional and acts as a pass counter. The <modifier> shown in the command description represents either a '>', used to mean 'after', and '<' to



represent 'before'. A count preceded by '>' will cause the breakpoint defined on the line to remain inactive until <count> number of times through that address. A count preceded by '<' will cause the breakpoint defined to be active for only the <count> number of times through that address, at which time it will be automatically removed. The <count> in either case must not exceed 32,000. The next part of the syntax is the optional <conditional>. This allows the breakpoint action to be dependent on some condition. The condition can be the contents of any machine register being equal or not equal to a hex value ('=' and '!=' respectively), or the contents of a specified memory location being zero or not zero. If a register is used, simply state the register name, followed by the relational, followed by the hex value (e.g. A=23, or B!=E2). To use a memory location, a dollar sign '\$' must precede the address. For example, \$100=0 would check if the byte at location hex 100 was zero, and \$A20!=0 would check if the byte at location hex A20 was not zero. If a memory address is specified, the only allowed value to the right of the relational is zero, and if any other value is used, it will be ignored. NOTE: The conditional part of the breakpoint definition may not be used with H,M, or J action codes. Two of the breakpoint actions require special syntax. These are the M (message) and J (jump) types. The M action is used to print a specified message to the terminal upon execution of the breakpoint. The J action is used to transfer control to another address (like a JMP instruction). Any breakpoint containing M may not contain J and vice versa. A breakpoint containing M should have an ASCII string following the <count> (or following the address if no count is specified). This string is the message which will be printed on the terminal each time the instruction is to be executed. Messages should be kept short (under 5 letters if possible). For the J type action, the hex address of the location of transfer should be provided after the <count> field. The examples below will help clarify the syntax.

## EXAMPLES:

B@100	Stop and print registers at \$100
B,SR@100	Same as above
B,H@A100	Set histogram at \$A100
B,ZR@300 >100	Zero states and print registers after 100 times through \$300
B@200, IF A=3C	Stop & print registers at \$200 only if acc. A = \$3C
B,M@210,SUB 1	Print message 'SUB 1' every time through location \$210

- continued -



B,J@100,1000      Transfer control to location \$1000  
                    when reach instruction at \$100  
B,TZ@400,<25,IF \$20=0  
                    For the first 25 times through  
                    location \$400, turn trace on and  
                    zero the states counter, but only  
                    if location \$20 is zero.

BP[,<address>[-<address>]]

PURPOSE:

The BP command is used to print the location of breakpoints and their associated action codes. The two address specifications are used to define the region of memory for checking breakpoints (beginning and ending, respectively). If no addresses are specified, all breakpoints will be listed. If only one address is given, then only the breakpoint at that address will be displayed (if one exists). Only the action codes are listed with each address.

EXAMPLES:

BP,10-C00      List breakpoints between \$10 & \$C00  
BP              List all breakpoints

CLB[,<address>[-<address>]]

PURPOSE:

Clear breakpoints in specified memory region. The addresses define the region of memory. If only one address is listed then only the breakpoint at that location will be cleared. If no addresses are specified, all breakpoints will be cleared.

EXAMPLES:

CLB              Clear all breakpoints  
CLB,0-100      Clear breakpoints between \$0 & \$100  
CLB,22A        Clear breakpoint at \$22A

CLH[,<address>[-<address>]]

PURPOSE:

Clear histogram counters in the specified memory region. The addresses define the region of memory. If only one address is listed then only the histogram counter at that location will be cleared. If no addresses are specified, all counters will be declared. NOTE: This command does not remove the histogram breakpoints, but clears its associated counter to zero in preparation for a new run.



## EXAMPLES:

CLH                      Clear all histogram counters  
 CLH,25-200            Clear counters between \$25 & \$200

## CLM

## PURPOSE:

Clear all messages in the breakpoint message table (used by the M action code, see the B command). This table is a fixed size and can be filled up. When deleting message type breakpoints using the CLB command, the associated space in the message table does not get freed. It is recommended that whenever all M type breakpoints have been cleared, also use the CLM command. Do not use this command if there are any active M type breakpoints. Their message strings will be destroyed!

## EXAMPLES:

CLM                      Clear all messages

## H[IST][,&lt;address&gt;[-&lt;address&gt;]]

## PURPOSE:

Print the histogram counter totals for the section of memory specified. The addresses define the region of memory. If only one address is listed then only the counter at that location is displayed. If no addresses are specified, all counter contents will be displayed. Each counter is shown preceded by its address. The counter value shows the number of times the instruction at that address has been executed.

## EXAMPLES:

HIST                      Display all histogram counters  
 H,0-200                  Display counters between 0 & \$200

## RTI,&lt;address&gt;]

## PURPOSE:

Start real time program execution (not simulated) at the current PC location. Program execution will halt at the <address> specified. This is similar to the standard breakpoint most users are familiar with in that memory is actually altered at the address specified (with a JMP instruction). Entering RT without an address will clear any real time breakpoint which may have been previously entered. This type of breakpoint and program execution is not recommended since no protection or checking is performed. When the program reaches the break address specified, the breakpoint is automatically cleared and the original code restored in memory. ROM may not be breakpointed with this command.



## EXAMPLES:

RT,600  
RT

Start at PC, end at \$600  
Clear an existing RT breakpoint

## VI. Memory Protection

The memory protection commands are a very powerful feature of the program debugger. The PROT command allows selected areas of memory to be write, execute, memory, or simulate protected. Write protected memory will cause a trap on any attempt to write to it. Execute protect will not allow opcodes to be fetched. Memory protect will not permit any type of reference; read, write, or execute. Simulate protect is used to protect sections of code which should not be simulated (executed in real time). It is important that only code called as a subroutine from non-simulate protected memory be contained in the area(s) of memory designated as simulate protected. An example would be to simulate protect the section of memory where a DOS resides. All subroutine calls to the DOS would then be executed in real time. Code which is simulate protected and does not follow this convention will usually cause the CPU to take over the execution of the program resulting in a loss of control. NOTE: To protect the memory around the machine stack (upper and lower bounds), use the 'memory' protection. This is the only type checked on stack references. Other commands in this group allow examination of protected memory regions or bounds, as well as the clearing of protection types.

BO[UNDS][,<types>]

## PURPOSE:

Display the bounds of protected memory. Each <type> specified will list all regions of memory protected by that type. <type> may be W, M, X, or S for write, memory, execute, and simulate, respectively. Multiple types may be displayed by listing the types on the command line separated by a comma or space. If no type is specified, all types of protection will be listed.

## EXAMPLES:

BOUNDS  
BO,M,X

Display all memory protection  
Display memory and execute  
protection bounds



CLP[,&lt;type&gt;]

## PURPOSE:

Clear all protected regions for a specified type of protection. The <type> is specified by the same letters described in BOUNDS. Only one type may be listed per command line. If type is not specified, all protection will be cleared.

## EXAMPLES:

CLP                      Clear all protection  
CLP,X                  Clear execute protection

PIROT],&lt;address&gt;[-&lt;address&gt;],&lt;type&gt;

## PURPOSE:

The PROT command is used to assign protection to a region of memory. The two <address> specifiers designate the beginning and ending addresses of the selected region. If only one address is specified, only the byte at that location will be protected. The <type> designator may either be M, X, W, or S for memory, execute, write, and simulate protection respectively. Only one type may appear with each address range. Multiple protection may be performed on one line by separating the range-type specifiers by a comma or a space.

## EXAMPLES:

PROT,0-100,M            Memory prot 0-\$100  
P,100,W,A100-A600,S    Write prot \$100 and simulate  
                         protect \$A100-A600

## VII. Execution Traps

Execution traps allow program stopping on certain general conditions. Several traps are always enabled. These include; trap on illegal opcode and trap on RTS if nest count=0. The user may enable and disable several other traps. These traps are for interrupt type instructions, transfer of address type instructions, trap on a selected subroutine depth (nest count), an instruction count timeout, and a general 'stop' address.

INST=&lt;count&gt;

## PURPOSE:

Set the instruction count timer to the value of count. If set to zero, this trap will be disabled. This timer is used to count the number of simulated instructions. Each time this counter reaches zero, the program will halt and print 'IC TIMEOUT AT XXXX', where XXXX is the address where the program stopped, and the counter will



be reset to the value it started at (the value specified by <count>).

EXAMPLES:

INST=400      Set counter to 400  
INST=0      Disable the instruction counter

IT[RAP]=ON or OFF

PURPOSE:

Turning the ITRAP on will cause the simulator to treat interrupt type instructions similar to illegal opcodes. Any time a RTI, SWI, or WAI instruction is found, the message 'I TRAP AT XXXX' will be displayed. The address of the instruction will be printed in place of the XXXX shown.

EXAMPLES:

ITRAP=ON      Enable the interrupt trap  
IT=OFF      Turn off the trap

NIEST=<value>

PURPOSE:

Set the nest trap at the level specified by <value>. The simulator will trap execution if a subroutine call instruction is found which will cause the nest level to equal or exceed that set by NEST. Setting the <value> to zero will disable this trap.

EXAMPLES:

NEST=6      Set nest trap to level 6  
N=0      Disable nest trap

STOP=<address>

PURPOSE:

The STOP trap is a general 'stop at address X' trap. It is useful for trapping returns to monitor type programs or operating systems. The trap is set at the address specified.

EXAMPLES:

STOP=100      Set stop trap at \$100  
STOP=E0D0      Set trap at MIKBUG entry



XFR=ON or OFF

PURPOSE:

Enabling the XFR trap will cause a trap each time a transfer of address type instruction is found (JMP, BRA, or BXX). This is useful for following major program flow. Typing a 'G' command after this trap will cause the program to start executing again.

EXAMPLES:

XFR=ON

Enable the transfer trap

XFR=OFF

Turn the trap off

## VIII. Interrupt Control

Both NMI and IRQ type interrupts may be simulated. Two modes of operation are possible. The first is automatic, periodic interrupt generation. This mode allows interrupts to be generated every N instructions. The second allows random interrupt generation from the keyboard. When these keys are typed during program simulation, the appropriate interrupt will be issued.

IRQ=<count>

PURPOSE:

Cause an IRQ type interrupt to be generated every <count> instructions. If count is set to zero, IRQ interrupts will be shut off.

EXAMPLES:

IRQ=5000

Generate IRQ every 5000 instructions

IRQ=0

Turn off automatic IRQs

NMI=<count>

PURPOSE:

Cause an NMI type interrupt to be generated every <count> instructions. If <count> is zero, automatic NMI interrupts will be turned off.

EXAMPLES:

NMI=300

Generate NMI every 300 instructions

NMI=0

Turn off automatic NMIs



'Control I'

PURPOSE:

Typing a 'control I' during program simulation will cause an IRQ type interrupt to be generated.

'Control N'

PURPOSE:

Typing a 'control N' during program simulation will cause an NMI type interrupt to be generated.



## Command Summary

## I. General System Control

C[ALC]  
DEL[AY]=<value>  
DEP[TH]  
E[XIT]  
F[LAG][=<address>]  
IND=ON or OFF  
MA[CH]  
MO[DE]=0 or 1  
R[EG]  
RES[ET]  
RET  
S[ET],<register list>  
STACK[,<value>]  
STAT[ES]  
TRAIL  
X,<o. s. command>

## II. Memory Commands

A[SM][,<address>]  
D[IS],<start address>,<stop address>  
DU[MP],<address>  
FIL[L],<start address>,<stop address>[,<byte>]  
FIN[D],<start address>,<stop address>,<string>  
M[EM],<address>

## III. Simulation Control

G[O]  
J[UMP],<address>  
PA[ST][,<value>]  
SIM[,<count>]  
ST[ART],<address>  
STEP[,<count>]  
T[RACE]=<value>  
TS[IM][,<count>]



#### IV. Breakpoints

B,<action>@<address>[,<modifier><count>][,IF<condition>]  
 B@<address>[,<modifier><count>][,IF<condition>]  
 BP[,<address>[-<address>]]  
 CLB[,<address>[-<address>]]  
 CLH[,<address>[-<address>]]  
 CLM  
 H[IST][, <address>[-<address>]]  
 RT[,<address>]

#### V. Memory Protection

BO[OUNDS][,<types>]  
 CLP[,<type>]  
 P[ROT],<address>[-<address>],<type>

#### VI. Execution Traps

INST=<count>  
 IT[RAP]=ON or OFF  
 N[EST]=<value>  
 STOP=<address>  
 XFR=ON or OFF

#### VII. Interrupt Control

IRQ=<count>  
 NMI=<count>



## Message Descriptions

The following is a list of all Debug generated messages and their respective meanings.

WHAT? = This is the general error message reported when an invalid input command has been entered.

"STOP" AT = The address set by the STOP trap command has been reached.

IC TIMEOUT AT = The number of instructions specified by the INST trap command have been executed.

ILLEGAL OPCODE AT = The instruction pointed to by the PC is an illegal opcode.

I TRAP AT = An SWI, RTI, or WAI instruction has been encountered and the ITRAP command has been used to enable the interrupt trap.

LAST XFR FROM = Displayed by request using the TRAIL command. The address gives the location of the last transfer of address type instruction which was executed.

SYNTAX ERROR = The command just entered does not follow the syntax rules for that command.

EP TRAP AT = An Execution Protect trap at the specified location resulting from an attempt to execute code in execute protected memory.

WP TRAP AT = A Write Protect trap at the specified location resulting from an attempt to write into write protected memory.

EX - MP TRAP AT = An attempt to execute code residing in memory protected memory has been detected at the specified address.

REF - MP TRAP AT = An attempt to reference (read or write) a byte in memory protected memory has been detected at the specified address.

SP TRAP AT = A Stack Pointer reference (PSH, JSR, etc.) was attempted in a section of memory which is memory protected.

TABLE OVERFLOW = The last command entered caused an internal table to overflow. The command does not get executed.

NC TRAP AT = A Nest Count trap occurred as a result of the nest level reaching the level specified in a NEST command.



RTS IN LEVEL 0 AT = An RTS instruction was encountered while the nest level was 0 (no previous call to subroutine had been executed).

NEST LEVEL IS 0 = There is no return address on the stack so the RET command can not display an address.

XFR TRAP AT = A transfer of address type instruction has been encountered with the transfer trap enabled (from XFR=ON).

MON XFR AT = The program being simulated tried to pass control to the monitor address which is used by the EXIT command.

OP HALT AT = An operator halt signal (control C character) was detected by the simulator.



## Getting Debug Running

The Debug Package loads from address \$3C00 through \$5FFF. The tape should be loaded into memory using your monitor system's load routine. Once loaded, the program to be debugged should be loaded. It is important that the two programs do not overlap in memory. If they do, consult the section of this manual on relocating the debugger. After all code is resident in memory, start the Debug Package at location \$4100, the cold start entry point. A '\*\*' prompt should appear. The cold start entry initializes all system tables, clears all registers, and clears out breakpoints. If it is necessary to re-enter the debugger after an EXIT command, the program should be entered at location \$4103, the warm start entry point. No clearing of values or tables is performed at this entry.







## Example Use

The following is an example debug session. It is assumed that the Debug Package is running and the program being tested is resident in memory. The sample program is shown first in its source listing form. Following is the sample debug operation.

## 1. Sample Program Source Listing

\*  
\* FIND THE MAX & MIN OF DATA LIST  
\*

0100                   ORG     \$0100

\* STORAGE LOCATIONS

0100           LARGE   RMB     1           LARGEST VALUE  
0101           SMALL   RMB     1           SMALLEST VALUE

0200                   ORG     \$0200

\* PROGRAM STARTS HERE

0200	CE 02 26	MINMAX	LDX	#DATA	POINT TO DATA STRING
0203	7F 01 00		CLR	LARGE	PRESET MAX
0206	86 FF		LDA A	#\$FF	ALSO
0208	B7 01 01		STA A	SMALL	PRESET MINIMUM
020B	A6 00	LOOP	LDA A	0,X	GET DATA ITEM
020D	B1 01 00		CMP A	LARGE	ITEM > LARGE ?
0210	24 03		BCC	CONT2	
0212	B7 01 00		STA A	LARGE	UPDATE LARGE
0215	B1 01 01	CONT2	CMP A	SMALL	ITEM < SMALL ?
0218	24 03		BCC	CONT3	
021A	B7 01 00		STA A	LARGE	UPDATE SMALL
021D	08	CONT3	INX		MOVE TO NEXT ITEM
021E	8C 02 2E		CPX	#DATEND	END OF LIST?
0221	26 E8		BNE	LOOP	IF NOT, REPEAT
0223	7E E0 D0		JMP	MON	RETURN TO MONITOR

\* DATA LIST

0226	02	DATA	FCB	2,54,76,32,12,87,55,6
022E		DATEND	EQU	*
E0D0		MON	EQU	\$E0D0     MONITOR EQUATE
			END	



## 11. Sample Debug Session

\*\*DIS,200,223

```

0200 LDX  #$0226
0203 CLR  $0100
0206 LDA  A  #$FF
0208 STA  A  $0101
020B LDA  A  0,X
020D CMP  A  $0100
0210 BCC  $0215
0212 STA  A  $0100
0215 CMP  A  $0101
0218 BCC  $021D
021A STA  A  $0100
021D INX
021E CPX  #$022E
0221 BNE  $020B
0223 JMP  $E0D0

```

DISASSEMBLE MACHINE CODE FROM  
\$200 TO \$223. SEE THE SOURCE  
LISTING FOR COMPARISON.

\*\*PROT,200,225,W

WRITE PROTECT THE PROGRAM AREA

\*\*BOUNDS,W

WRITE PROTECTION

DISPLAY THE PROTECTION BOUNDS.

0200-0225

\*\*R

DISPLAY THE REGISTERS

C=00 A=00 B=00 X=0000 S=A07F P=0000 N=00 0000 ADC A \$B9B9

\*\*START,200 START PROGRAM AT \$200

MON XFR AT E0D0 - MONITOR TRANSFER TRAP.

\*\*M,100

U100 06 . EXAMINE \$100 &amp; \$101 (LARGE &amp; SMALL)

0101 FF - RESULT IS NOT CORRECT!

\*\*SET P=200 SET PC &amp; EXAMINE REGISTERS

\*\*R

C=C5 A=06 B=00 X=022E S=A07F P=0200 N=00 0200 LDX #\$0226

\*\*IND=ON SET IND &amp; FLAG, NOTE RESULT

\*\*FLAG=100

\*\*R

C=C5 A=06 B=00 X=022E S=A07F P=0200 N=00 I=B9 F=06FF 0200 LDX #\$0226

\*\*TSIM,10 TRACE 10 INSTRUCTIONS

C=C1 A=06 B=00 X=0226 S=A07F P=0203 N=00 I=02 F=06FF 0203 CLR \$0100

C=C4 A=06 B=00 X=0226 S=A07F P=0206 N=00 I=02 F=00FF 0206 LDA A #\$FF

C=C8 A=FF B=00 X=0226 S=A07F P=0208 N=00 I=02 F=00FF 0208 STA A \$0101

C=C8 A=FF B=00 X=0226 S=A07F P=020B N=00 I=02 F=00FF 020B LDA A 0,X

C=C0 A=02 B=00 X=0226 S=A07F P=020D N=00 I=02 F=00FF 020D CMP A \$0100

C=C0 A=02 B=00 X=0226 S=A07F P=0210 N=00 I=02 F=00FF 0210 BCC \$0215

C=C0 A=02 B=00 X=0226 S=A07F P=0215 N=00 I=02 F=00FF 0215 CMP A \$0101

C=C1 A=02 B=00 X=0226 S=A07F P=0218 N=00 I=02 F=00FF 0218 BCC \$021D

C=C1 A=02 B=00 X=0226 S=A07F P=021A N=00 I=02 F=00FF 021A STA A \$0100

C=C1 A=02 B=00 X=0226 S=A07F P=021D N=00 I=02 F=02FF 021D INX

\*\*B@218

SET 'SR' BREAKPOINT AT \$218

\*\*BP

0218 - SR

DISPLAY ALL BREAKPOINTS.



START PROGRAM AT PC.

\*\*G

HIT BREAKPOINT →

C=C1 A=36 B=00 X=0227 S=A07F P=0218 N=00 I=36 F=02FF 0218 BCC \$021D

\*\*TSIM - TRACE 1 INSTRUCTION.

C=C1 A=36 B=00 X=0227 S=A07F P=021A N=00 I=36 F=02FF 021A STA A \$0100

\*\*ASM, 21A

U21A STAA \$101 USE ASM TO FIX !

Should be STA A \$101!

U21D

\*\*CLB

CLEAR ALL BREAKPOINTS.

\*\*START 200

RUN PROGRAM AGAIN

MON XFR AT E0D0

\*\*M 100

U100 00 .

EXAMINE LARGE &amp; SMALL

0101 02

SMALL IS OK! LARGE IS STILL WRONG.

\*\*TRACE=40

\*\*START, 200 ENABLE TRACE TO 40 &amp; RUN.

C=C0 A=06 B=00 X=0226 S=A07F P=0203 N=00 I=02 F=0002 0203 CLR \$0100  
 C=C4 A=06 B=00 X=0226 S=A07F P=0206 N=00 I=02 F=0002 0206 LDA A #\$FF  
 C=C8 A=FF B=00 X=0226 S=A07F P=0208 N=00 I=02 F=0002 0208 STA A \$0101  
 C=C8 A=FF B=00 X=0226 S=A07F P=020B N=00 I=02 F=00FF 020B LDA A 0,X  
 C=C0 A=02 B=00 X=0226 S=A07F P=020D N=00 I=02 F=00FF 020D CMP A \$0100  
 C=C0 A=02 B=00 X=0226 S=A07F P=0210 N=00 I=02 F=00FF 0210 BCC \$0215  
 C=C0 A=02 B=00 X=0226 S=A07F P=0215 N=00 I=02 F=00FF 0215 CMP A \$0101  
 C=C1 A=02 B=00 X=0226 S=A07F P=0218 N=00 I=02 F=00FF 0218 BCC \$021D  
 C=C1 A=02 B=00 X=0226 S=A07F P=021A N=00 I=02 F=00FF 021A STA A \$0101  
 C=C1 A=02 B=00 X=0226 S=A07F P=021D N=00 I=02 F=0002 021D INX  
 C=C1 A=02 B=00 X=0227 S=A07F P=021E N=00 I=36 F=0002 021E CPX #\$022E  
 C=C1 A=02 B=00 X=0227 S=A07F P=0221 N=00 I=36 F=0002 0221 BNE \$020B  
 C=C1 A=02 B=00 X=0227 S=A07F P=020B N=00 I=36 F=0002 020B LDA A 0,X  
 C=C1 A=36 B=00 X=0227 S=A07F P=020D N=00 I=36 F=0002 020D CMP A \$0100  
 C=C0 A=36 B=00 X=0227 S=A07F P=0210 N=00 I=36 F=0002 0210 BCC \$0215  
 C=C0 A=36 B=00 X=0227 S=A07F P=0215 N=00 I=36 F=0002 0215 CMP A \$0101  
 C=C0 A=36 B=00 X=0227 S=A07F P=0218 N=00 I=36 F=0002 0218 BCC \$021D  
 C=C0 A=36 B=00 X=0227 S=A07F P=021D N=00 I=36 F=0002 021D INX

OP HALT AT 021D

\*\*DIS 20B 210

SHOULD NOT HAVE BRANCHED

020B LDA A 0,X

020D CMP A \$0100

0210 BCC \$0215

SHOULD BE 'BHS' INSTRUCTION!

\*\*ASM 210

U210 BLS \$215

USE ASM TO CORRECT CODE.

U212

\*\*T=0

\*\*START 200

SET TRACE TO 0 (off) AND RUN PROGRAM.

MON XFR AT E0D0

\*\*M 100

0100 67 .

ANSWERS ARE NOW CORRECT!

0101 02



\*\*B HQ200  
 \*\*B HQ20B  
 \*\*B HQ215  
 \*\*B HQ21D  
 \*\*BP

*PROFILE THE PROGRAM WITH HISTOGRAM  
 BREAKPOINTS AT \$200, 20B, 215, & 21D.*

0200 - H  
 020B - H  
 0215 - H  
 021D - H

*DISPLAY ALL BREAKPOINTS.*

\*\*START 200  
 MON XFR AT EOD0  
 \*\*HIST

*RUN PROGRAM - 'START' CLEARS  
 THE STATES COUNTER.*

0200 - 0  
 020B - 8  
 0215 - 8  
 021D - 8

*HISTOGRAM PRINTOUT*

\*\*STATES  
 STATES = 00000300  
 \*\*DIS 200 223

*PROGRAM REQUIRED 300 MACHINE CYCLES.*

0200 LDX #\$0226  
 0203 CLR \$0100  
 0206 LDA A #\$FF  
 0208 STA A \$0101  
 020B LDA A 0,X  
 020D CMP A \$0100  
 0210 BLS \$0215  
 0212 STA A \$0100  
 0215 CMP A \$0101  
 0218 BCC \$021D  
 021A STA A \$0101  
 021D INX  
 021E CPX #\$022E  
 0221 BNE \$020B  
 0223 JMP \$E0D0

*DISASSEMBLE FINAL PROGRAM.*

\*\*EXIT  
 \$

*EXIT THE DEBUGGER.*



## Adapting to Your System

The following descriptions may prove helpful in adapting this program to non-standard systems. All I/O and stack references are described below.

## I. I/O References

GETCHR at \$4106. This jump vector references the standard input character routine in the SWTBUG monitor ROM. Any input routine may be used as long as it returns the ASCII character in the A accumulator with the parity removed, and preserves the B and X registers.

PUTCHR at \$4109. This jump vector references the standard output character routine in the SWTBUG monitor ROM. Any output routine may be used as long as it outputs the character from the A accumulator, and preserves the B and X registers.

WARMS at \$410C. This jump vector references the starting entry address of the SWTBUG monitor ROM. This may be changed to the starting address of your own monitor. This is the address used by the EXIT command.

## II. I/O Related Storage

ACIA at \$410F. This FDB formed address is a pointer to the ACIA base address used by the basic input and output routines. Change as needed. NOTE: The Debug Package requires an ACIA type serial interface to function correctly.

BSP at \$4111. This byte contains the character which is decoded as the backspace character (currently a Control H, \$08). Change as desired.

DEL at \$4112. This byte contains the character which is decoded as the line cancel character (currently a Control X, \$18). Change as desired.

BSE at \$4113. This byte contains the character which will be echoed after the receipt of a backspace character (currently a Control H, \$08). If this character is set to \$08, a space will be output preceding the backspace echo character. Setting this byte to zero will inhibit the backspace echo character.

ESC at \$4114. This byte contains the character which is decoded as the Escape character (currently an ASCII Escape, \$1B). This may be changed as desired.



### III. Stack Pointer References

Load Stack at \$411B and \$4195. These two locations contain LDS instructions and set the stack to \$3FFF. They may be changed as desired.

### IV. The X Command

The X command calls a section of code at location \$5589. This is implemented for the FLEX disk operating system and calls FLEX to perform a specified command. If you are using a different operating system, you may substitute your own code to perform the equivalent. The code may reside from \$5589 through \$55A2.

### V. System Tables

The Debug Package uses several system tables which reside from \$3C00 to \$3F9F. They are named and sized as follows:

BPTAB	RMB	256	Allows	32	breakpoints
STRTPC	RMB	512	"	256	past instructions
SMTAB	RMB	32	"	8	sim prot fields
EXTAB	RMB	32	"	8	ex prot fields
WPTAB	RMB	32	"	8	write prot fields
MTAB	RMB	32	"	8	mem prot fields
MSGTAB	RMB	32	"	approx	5 messages

These tables may be moved and expanded to allow more breakpoints and protection fields as desired. Complete details will not be given here, as this is a job for the more experienced programmer.

### VI. Saving the Altered Program

After modifications have been made to the program, it may be saved on mass storage. The program should be saved from \$4100 through \$5FFF. The starting or transfer address is \$4100.



## Relocating the Debug Package

The Debug Package may be relocated in memory by using the TSC 6800 Relocator (part number SL68-28). The Debug Package as sold resides from \$3C00 to \$5FFF. It may be moved easily to any lower memory location and to any location higher than \$5C00. The example below shows relocation to \$5C00 which moves the cold start entry address to \$6100 (from \$4100). The relocated version will reside from \$5C00 to \$7FFF. If it is necessary to move the program to an area between \$3C00 and \$5C00, two relocations must be performed, one moving it to a lower location, and then up to the desired position. This is necessary because of program overlap. NOTE: The Debug Package must always start on a page boundary.

## Relocation Example

\* TSC 6800 RELOCATOR \*

PRESENT PROGRAM:

BEGIN ADDRESS? 4100

END ADDRESS? 5FFF

MOVE TO? 6100

FIX REFERENCES? Y

LOAD FROM TAPE? N

DATA BLOCKS? Y

BEGIN ADDRESS? 410F

END ADDRESS? 411A

BEGIN ADDRESS? 57A8

END ADDRESS? 5FFF

BEGIN ADDRESS? FFFF

ALTER RANGE? Y

BEGIN ADDRESS? 3B00

END ADDRESS? 5FFF

FIX FDB'S? Y

ADDRESS? 57AC

ADDRESS? 57B0

" 57B9

" 57BE

" 57C5

" 57CB

" 57D1

" 57D7

" 57DD

" 57E3

" 57EB

" 57F3

" 57FA

" 5801

" 5808

" 580F

" 5816



" 581B  
 " 5822  
 " 5828  
 " 582F  
 " 5835  
 " 583D  
 " 5844  
 " 584A  
 " 5851  
 " 5858  
 " 585F  
 " 5865  
 " 586C  
 " 5873  
 " 5879  
 " 5881  
 " 5887  
 " 588C  
 " 5892  
 " 5898  
 " 58A0  
 " 58A8  
 " 58B1  
 " 58B8  
 " 58BF  
 " 58C7  
 " 58CF  
 " 58D6  
 " 58DA  
 " 58E0  
 " 58EB  
 " 58ED  
 " 58EF  
 " 58F1  
 " 58F3  
 " 58F5  
 " 58F7  
 " 58F9  
 " 58FB  
 " 58FD  
 " 58FF  
 " 5901  
 " 5903  
 " 5905  
 " 5909  
 " 590D  
 " 5911  
 " 5915  
 " 5919  
 " 591D

ADDRESS? 5921  
 ADDRESS? FFFF

RELOCATION COMPLETE !!!



Source Listing



*E. J. Connelley*



and the world



Normal 3



\*  
 \* TSC DEBUG PACKAGE  
 \*  
 \* COPYRIGHT (C) 1978 BY  
 \* TECHNICAL SYSTEMS CONSULTANTS, INC.  
 \* BOX 2574  
 \* W. LAFAYETTE, INDIANA 47906  
 \* (317) 423-5465  
 \*

\* TEMP STORAGE

4000		ORG	\$4000
4000	LINBUF	RMB	80
4050 00 00	BUFPT	FDB	0
4052 00 00	CRSAVE	FDB	0
4054 00 00	INDEX	FDB	0
4056 00 00	DATPNT	FDB	0
4058 00 00	XSAVE	FDB	0
405A 00	OUTNUM	FCB	0
405B 00	TEMP	FCB	0
405C 00	TEMP1	FCB	0
405D 00 00	VALUE	FDB	0
405F 00 00	POINT	FDB	0
4061 00 00	POINT2	FDB	0
4063 00 00	FIRST	FDB	0
4065 00 00	LAST	FDB	0
4067 00	CNTR	FCB	0
4068 00	LSTTRM	FCB	0
4069 00	RTBF	FCB	0
406A 00 00	RTAD	FDB	0
406C 00	RTDAT	FCB	0, 0, 0
406F 00	MOD	FCB	0
4070 00	OPND	FCB	0, 0, 0
4073 00 00	END	FDB	0
4075 00 00	COUNT	FDB	0
4077 00 00	MAXC	FDB	0
4079 00 00	NMIC	FDB	0
407B 00 00	NMIC2	FDB	0
407D 00 00	IRQC	FDB	0
407F 00 00	IRQC2	FDB	0
4081 00 00	MAXSP	FDB	0
4083 00 00	TSP	FDB	0
4085 00 00	STPCNT	FDB	0
4087 00 00	SIMCNT	FDB	0
4089 00	WAITF	FCB	0
408A 00	PAUSF	FCB	0
408B 00	DELAY	FCB	0
408C 00	CC	FCB	0
408D 00	AR	FCB	0
408E 00	BR	FCB	0
408F 00 00	XR	FDB	0



4091	00	00	SP	FDB	0
4093	00	00	PC	FDB	0
4095	00	00	SSP	FDB	0
4097	00	00	OLDPC	FDB	0
4099	00		NESTC	FCB	0
409A	00		OPCNT	FCB	0
409B	00		MODE	FCB	0
409C	00		PRON	FCB	0
409D	00		TRCF	FCB	0
409E	00	00	FLAGA	FDB	0
40A0	00		FLAGB	FCB	0
40A1	00		VFLG	FCB	0
40A2	00		EAFL	FCB	0
40A3	00		NOBKF	FCB	0
40A4	00		CFLG	FCB	0
40A5	00		NSTRP	FCB	0
40A6	00		XFRT	FCB	0
40A7	00	00	STATES	FDB	0, 0
40AB	00		ITRF	FCB	0
40AC	00	00	BPPOS	FDB	0
40AE	00	00	BPEND	FDB	0
40B0	00		BPCODE	FCB	0
40B1	00		BTYP	FCB	0
40B2	00	00	BPAD	FDB	0
40B4	00	00	NXTBP	FDB	0
40B6	00	00	NXTMSG	FDB	0
40B8	00		INCOD	FCB	0
40B9	00		EQUALS	FCB	0
40BA	00	00	RELADR	FDB	0
40BC	00	00	MARKRG	FDB	0
40BE	00	00	LASTJ	FDB	0
40C0	00	00	NXTPC	FDB	0
40C2	00	00	OPPNT	FDB	0
40C4	00	00	OPPNT2	FDB	0
40C6	00	00	SMEND	FDB	0
40C8	00	00	EXEND	FDB	0
40CA	00	00	WPEND	FDB	0
40CC	00	00	MEND	FDB	0
40CE	00	00	TABE	FDB	0
40D0	00	00	EADR	FDB	0
40D2	00	00	POPTAB	FDB	0
40D4	00	00	OP	FDB	0
40D6	00			FCB	0
40D7	00	00	OPJMP	FDB	0
40D9	00			FCB	0
40DA	00	00	BXOP	FDB	0
40DC	00			FCB	0
40DD	00	00	BXOP2	FDB	0
40DF	00			FCB	0

\*  
\* SYSTEM EQUATES  
\*



0007	BELL	EQU	\$07	BELL CHARACTER (1G)
000D	CR	EQU	\$0D	CARRIAGE RETURN
000A	LF	EQU	\$0A	LINE FEED
0020	SPC	EQU	\$20	SPACE
3FFF	STACK	EQU	\$3FFF	

4100 ORG \$4100

\*  
\* SYSTEM ENTRY  
\*

4100	7E 41 1B	COLDS	JMP	COLD
4103	7E 41 95	WARMST	JMP	WARM

\* SYSTEM CONSTANTS AND JUMP VECTORS

4106	7E E1 AC	GETCHR	JMP	\$E1AC	BD E440
4109	7E E1 D1	PUTCHR	JMP	\$E1D1	7E E44C
410C	7E E0 D0	WARMS	JMP	\$E0D0	
410F	80 04	ACIA	FDB	\$8004	
4111	08	BSP	FCB	\$08	
4112	18	DEL	FCB	\$18	
4113	08	BSE	FCB	\$08	
4114	1B	ESC	FCB	\$1B	
4115	FF FA	SWIV	FDB	\$FFFA	
4117	FF F8	IRQV	FDB	\$FFF8	
4119	FF FC	NMIV	FDB	\$FFFC	
4107		INV	EQU	GETCHR+1	
410A		OUTV	EQU	PUTCHR+1	
410D		MONV	EQU	WARMS+1	

\*  
\* COLD START ENTRY  
\*

411B	8E 3F FF	COLD	LDS	#STACK	
411E	CE 40 63		LDX	#FIRST	POINT TO TEMPS
4121	6F 00	COLD2	CLR	0, X	CLEAR STORAGE
4123	08		INX		
4124	8C 40 D0		CPX	#EADR	END OF AREA?
4127	26 F8		BNE	COLD2	
4129	CE 47 1A		LDX	#TRCRET	SET TEMP CODE
412C	FF 40 D8		STX	OPJMP+1	
412F	86 7E		LDA A	#7E	SET JUMP
4131	B7 40 D7		STA A	OPJMP	
4134	B7 40 DD		STA A	BXOP2	
4137	CE 47 3F		LDX	#PBRA	SET CON BRA
413A	FF 40 DE		STX	BXOP2+1	
413D	86 39		LDA A	#39	SET RETURN
413F	B7 40 DC		STA A	BXOP+2	



```

4142 86 01      LDA A  #1      SET REL BRA
4144 B7 40 DB    STA A  BXOP+1
4147 CE 5B 00    LDX      #OPTAB  SET TABLE
414A FF 40 D2    STX      POPTAB
414D FF 40 C2    STX      OPPNT
4150 FF 40 C4    STX      OPPNT2
4153 CE 5F 00    LDX      #AUXTAB  SET POINTER
4156 FF 40 C4    STX      OPPNT2
4159 CE A0 7F   LDX      ##A07F  SET INITIAL STACK
415C FF 40 91    STX      SP
415F FF 40 81    STX      MAXSP
4162 CE 3C 00    LDX      #BPTAB
4165 FF 40 AE    STX      BPEND
4168 CE 3D 00    LDX      #STRTPC  SET UP TABLES
416B FF 40 C0    STX      NXTPC
416E CE 3F 80    LDX      #MSGTB  SET TABLE
4171 FF 40 B6    STX      NXTMSG
4174 8D 06      BSR      CLRPR  CLEAR PR TABLES
4176 7C 40 9B    INC      MODE   SET MODE TO 1
4179 7E 41 95    JMP      WARM

```

```

*
* CLEAR PROTECTION TABLES
*

```

```

417C CE 3F 20    CLRPR  LDX      #EXTAB  SET ALL EMPTY
417F FF 40 C8    STX      EXEND
4182 CE 3F 00    LDX      #SMTAB
4185 FF 40 C6    STX      SMEND
4188 CE 3F 40    LDX      #WPTAB
418B FF 40 CA    STX      WPEND
418E CE 3F 60    LDX      #MTAB
4191 FF 40 CC    STX      MEND
4194 39          RTS

```

```

*
* WARM START ENTRY
*

```

```

4195      WARM      EQU      *

```

```

* EXECUTIVE STARTS HERE

```

```

4195 8E 3F FF    EXEC    LDS      #STACK  SETUP STACK
4198 7F 40 8A    CLR      PAUSE  CLEAR PAUSE
419B 86 01      LDA A  #1      SET NO BREAK
419D B7 40 A3    STA A  NOBK
41A0 B6 40 9D    LDA A  TRCF  SET TRACE
41A3 B7 40 9C    STA A  PRON
41A6 CE 59 35    LDX      #PRMPT  POINT TO PROMPT
41A9 BD 42 85    JSR      PSTRNG  OUTPUT IT
41AC BD 42 3C    JSR      INBUF  GET INPUT LINE
41AF BD 43 7E    EXEC3    JSR      SKPSPC  SKIP SPACES

```



41B2 81 0D		CMP A	#CR	LONE CARRIAGE RET?
41B4 27 DF		BEQ	EXEC	
41B6 BD 43 42		JSR	CLASS	CLASSIFY CHAR
41B9 25 0B		BCS	ILIN	ERROR?
41BB CE 57 A8		LDX	#COMTBL	POINT TO TABLE
41BE 8D 13		BSR	LKNAM	LOOK FOR NAME
41C0 26 04		BNE	ILIN	ERROR?
41C2 EE 01	EXEC6	LDX	1,X	GET COM ADDRESS
41C4 6E 00		JMP	0,X	JUMP TO IT

\*  
\* REPORT ILLEGAL INPUT  
\*

41C6 CE 59 3B	ILIN	LDX	#WHATST	POINT TO STRING
41C9 BD 42 85	ILIN2	JSR	PSTRNG	OUTPUT IT
41CC 20 C7	ILIN4	BRA	EXEC	RESTART

\*  
\* REPORT SYNTAX ERROR  
\*

41CE CE 59 F0	SYNER	LDX	#SYNST	POINT TO STRING
41D1 20 F6		BRA	ILIN2	

\*  
\* LOOK FOR NAME IN TABLE  
\*

41D3 FF 40 54	LKNAM	STX	INDEX	SAVE POINTER
41D6 FE 40 50	LKNAM2	LDX	BUFPNT	SET POINTER
41D9 A6 00	LKNAM3	LDA A	0,X	GET A CHARACTER
41DB BD 43 42		JSR	CLASS	CLASSIFY IT
41DE 25 4B		BCS	LKNAM9	TERM?
41E0 81 5F		CMP A	#\$5F	CHECK IF UPPER CASE
41E2 23 02		BLS	LKNAM4	IT IS
41E4 80 20		SUB A	#\$20	MAKE UPPER
41E6 FF 40 56	LKNAM4	STX	DATPNT	SAVE INDEX
41E9 FE 40 54		LDX	INDEX	RESTORE POINTER
41EC A1 00		CMP A	0,X	CHECK CHARACTER
41EE 26 2B		BNE	LKNAM6	
41F0 08		INX		BUMP THE POINTER
41F1 FF 40 54		STX	INDEX	SAVE INDEX
41F4 6D 00		TST	0,X	CHECK IF END
41F6 26 1D		BNE	LKNAM5	
41F8 FE 40 56	LKNA41	LDX	DATPNT	POINT TO NAME
41FB 08		INX		BUMP TO NEXT
41FC A6 00		LDA A	0,X	GET CHARACTER
41FE BD 43 42		JSR	CLASS	CLASSIFY IT
4201 24 0D		BCC	LKNA45	TERM?
4203 81 0D		CMP A	#CR	IS IT RETURN?
4205 27 01		BEQ	LKNA42	
4207 08		INX		BUMP TO NEXT
4208 FF 40 50	LKNA42	STX	BUFPNT	SAVE NEW POSITION



420B FE 40 54		LDX	INDEX	POINT TO TABLE
420E 4F		CLR A		SHOW FOUND
420F 39		RTS		RETURN
4210 FE 40 54	LKNA45	LDX	INDEX	RESET POINTER
4213 20 07		BRA	LKNAM7	CONTINUE
4215 FE 40 56	LKNAM5	LDX	DATPNT	RESET POINTER
4218 08		INX		BUMP THE POINTER
4219 20 BE		BRA	LKNAM3	
421B 08	LKNAM6	INX		BUMP POINTER
421C 6D 00	LKNAM7	TST	0,X	END OF WORD?
421E 26 FB		BNE	LKNAM6	REPEAT
4220 08		INX		BUMP PAST ADDRESS
4221 08		INX		
4222 08		INX		
4223 6D 00		TST	0,X	END OF TABLE?
4225 26 AC		BNE	LKNAM	REPEAT IF NOT
4227 8C 00 00		CPX	#0000	SET NE BIT
422A 39	LKNAM8	RTS		RETURN
422B 09	LKNAM9	DEX		SAVE POINTER
422C FF 40 56		STX	DATPNT	
422F FE 40 54	LKNA95	LDX	INDEX	POINT TO TABLE
4232 6D 00		TST	0,X	END OF LIST?
4234 27 C2		BEQ	LKNA41	
4236 08		INX		BUMP TO NEXT
4237 FF 40 54		STX	INDEX	
423A 20 F3		BRA	LKNA95	REPEAT

\*

\* INPUT LINE INTO BUFFER

\*

423C CE 40 00	INBUF	LDX	#LINBUF	POINT TO BUFFER
423F FF 40 50		STX	BUFPNT	SET POINTER
4242 BD 41 06	INBUF2	JSR	GETCHR	GO GET A CHARACTER
4245 B1 41 12		CMP A	DEL	IS IT DELETE?
4248 27 1A		BEQ	INBUF5	
424A B1 41 11		CMP A	BSP	IS IT BACK SPACE?
424D 27 1C		BEQ	INBUF6	
424F 81 0D		CMP A	#CR	IS IT CARRIAGE RET?
4251 27 09		BEQ	INBUF4	
4253 81 1F		CMP A	##1F	IS IT CONTROL?
4255 23 EB		BLS	INBUF2	IGNORE IF SO
4257 8C 40 4F	INBUF3	CPX	#LINBUF+79	
425A 27 E6		BEQ	INBUF2	
425C A7 00	INBUF4	STA A	0,X	PUT CHAR IN BUFFER
425E 08		INX		BUMP THE POINTER
425F 81 0D		CMP A	#CR	IS IT RETURN?
4261 26 DF		BNE	INBUF2	REPEAT IF NOT
4263 39		RTS		RETURN
4264 CE 59 96	INBUF5	LDX	#DELST	POINT TO STRING
4267 8D 1C		BSR	PSTRNG	OUTPUT IT
4269 20 D1		BRA	INBUF	
426B 8C 40 00	INBUF6	CPX	#LINBUF	FRONT OF BUFFER?
426E 27 F4		BEQ	INBUF5	



4270 09	DEX	DEC THE POINTER
4271 B6 41 13	LDA A BSE	GET ECHO CHAR
4274 81 08	CMP A #8	IS IT 1H ?
4276 26 08	BNE INBU65	
4278 86 20	LDA A #SPC	SETUP SPACE
427A BD 41 09	JSR PUTCHR	OUTPUT IT
427D B6 41 13	LDA A BSE	GET CHAR
4280 BD 41 09	INBU65 JSR PUTCHR	OUTPUT IT
4283 20 BD	BRA INBUF2	REPEAT

\*  
\* PRINT STRING AT X  
\*

4285 8D 0D	PSTRNG BSR PCRLF	OUTPUT CR & LF
4287 A6 00	PDATA1 LDA A 0,X	GET A CHARACTER
4289 81 04	CMP A #4	IS IT TERM?
428B 27 06	BEQ PDATA2	
428D BD 41 09	JSR PUTCHR	GO PUT CHAR.
4290 08	INX	BUMP THE POINTER
4291 20 F4	BRA PDATA1	REPEAT IT
4293 39	PDATA2 RTS	

\*  
\* PCRLF ROUTINE  
\*

4294 FF 40 52	PCRLF STX CRSAVE	SAVE X
4297 7D 40 8A	TST PAUSF	PAUSE?
429A 26 1A	BNE WAITR	
429C FE 41 0F	LDX ACIA	POINT TO ACIA
429F A6 00	LDA A 0,X	GET STATUS
42A1 46	ROR A	CHARACTER?
42A2 24 09	BCC PCRLF2	
42A4 A6 01	LDA A 1,X	GET CHARACTER
42A6 84 7F	AND A #\$7F	MASK PARITY
42A8 B1 41 14	CMP A ESC	IS IT ESCAPE?
42AB 27 09	BEQ WAITR	
42AD CE 59 99	PCRLF2 LDX #CRLFST	POINT TO STRING
42B0 8D 05	BSR PDATA1	PRINT IT
42B2 FE 40 52	LDX CRSAVE	RESTORE X
42B5 39	RTS	RETURN

\*  
\* WAIT FOR RESPONSE  
\*

42B6 FE 41 0F	WAITR LDX ACIA	POINT TO ACIA
42B9 7F 40 8A	CLR PAUSF	CLEAR FLAG
42BC A6 00	WAITR1 LDA A 0,X	GET STATUS
42BE 46	ROR A	CHARACTER?
42BF 24 FB	BCC WAITR1	
42C1 A6 01	LDA A 1,X	GET CHARACTER
42C3 84 7F	AND A #\$7F	MASK PARITY



42C5 B1 41 14	CMP A	ESC	IS IT ESCAPE?
42C8 27 E3	BEQ	PCRLF2	
42CA 81 03	CMP A	##3	IS IT 1C?
42CC 27 04	BEQ	WAITR2	
42CE 81 0D	CMP A	##D	IS IT CR?
42D0 26 EA	BNE	WAITR1	
42D2 7E 41 95	WAITR2	JMP EXEC	RETURN TO EXEC

\*

\* OUTPUT DECIMAL NUMBER AT X

\*

42D5 7F 40 5A	OUTDEC	CLR	OUTNUM	CLEAR FLAG
42D8 F7 40 5C		STA B	TEMP1	SET SUP FLAG
42DB 86 04	OUTDE2	LDA A	#4	SET COUNTER
42DD B7 40 5B		STA A	TEMP	SAVE IT
42E0 A6 00		LDA A	0, X	GET MSB
42E2 E6 01		LDA B	1, X	GET LSB
42E4 CE 58 E3		LDX	#CONTRBL	POINT TO CONSTANTS
42E7 8D 0A	OUTDE4	BSR	OUTDIG	OUTPUT DIGIT
42E9 08		INX		BUMP TO NEXT CONST.
42EA 08		INX		
42EB 7A 40 5B		DEC	TEMP	DEC THE COUNT
42EE 26 F7		BNE	OUTDE4	
42F0 17		TBA		GET LS DIGIT
42F1 20 42		BRA	OUTHR	OUTPUT IT

\*

\* OUTPUT DECIMAL DIGIT

\*

42F3 7F 40 75	OUTDIG	CLR	COUNT	CLEAR COUNTER
42F6 A1 00	OUTDI2	CMP A	0, X	CHECK MSB
42F8 25 0F		BLO	OUTDI5	
42FA 22 04		BHI	OUTDI4	
42FC E1 01		CMP B	1, X	COMPARE LSB
42FE 25 09		BLO	OUTDI5	
4300 E0 01	OUTDI4	SUB B	1, X	SUB LSB
4302 A2 00		SBC A	0, X	SUB MSB
4304 7C 40 75		INC	COUNT	BUMP COUNTER
4307 20 ED		BRA	OUTDI2	REPEAT
4309 36	OUTDI5	PSH A		SAVE A
430A B6 40 75		LDA A	COUNT	GET TOTAL
430D 26 10		BNE	OUTDI6	IS IT ZERO?
430F 7D 40 5A		TST	OUTNUM	SUPRESS ZEROES?
4312 26 0B		BNE	OUTDI6	NUMBER YET?
4314 7D 40 5C		TST	TEMP1	NULL OR SPACE?
4317 27 0B		BEQ	OUTDI8	
4319 86 20		LDA A	#SPC	SETUP SPACE
431B 8D 22		BSR	OUTHR2	OUTPUT IT
431D 20 05		BRA	OUTDI8	
431F 7C 40 5A	OUTDI6	INC	OUTNUM	SHOW NUMBER
4322 8D 11		BSR	OUTHR	OUTPUT DIGIT
4324 32	OUTDI8	PUL A		RESTORE A



4325 39

RTS

RETURN

\*

\* OUTPUT FOUR HEX DIGITS AT X

\*

4326 8D 01

OUTADR

BSR

OUTHEX

OUT 2 DIGITS

4328 08

INX

BUMP POINTER

\*

\* OUTPUT HEX DIGIT AT X

\*

4329 A6 00

OUTHEX

LDA A

0,X

GET MSB

432B 8D 04

BSR

OUTHL

OUTPUT IT

432D A6 00

LDA A

0,X

DO LSB

432F 20 04

BRA

OUTHR

OUTPUT IT

4331 44

OUTHL

LSR A

GET MSB TO LSB

4332 44

LSR A

4333 44

LSR A

4334 44

LSR A

4335 84 0F

OUTHR

AND A

#\$F

MASK OFF MSB

4337 8B 30

ADD A

#\$30

ADD IN BIAS

4339 81 39

CMP A

#'9

OVER NUMBERS?

433B 23 02

BLS

OUTHR2

433D 8B 07

ADD A

#7

FINISH BIAS

433F 7E 41 09

OUTHR2

JMP

PUTCHR

OUTPUT IT

\*

\* CLASSIFY CHARACTER

\*

4342 81 30

CLASS

CMP A

#'0

IS IT 0?

4344 25 14

BLO

CLASS2

REPORT

4346 81 39

CMP A

#'9

COMPARE TO 9

4348 23 15

BLS

CLASS4

IS IT NUMBER?

434A 81 41

CMP A

#'A

COMPARE TO A

434C 25 0C

BLO

CLASS2

REPORT

434E 81 5A

CMP A

#'Z

COMPARE TO Z

4350 23 0D

BLS

CLASS4

IS IT LETTER?

4352 81 61

CMP A

#'A+\$20

CHECK FOR LOWER

4354 25 04

BLO

CLASS2

REPORT

4356 81 7A

CMP A

#'Z+\$20

UPPER LIMIT

4358 23 05

BLS

CLASS4

435A 0D

CLASS2

SEC

SET FOR NOT

435B B7 40 68

STA A

LSTTRM

435E 39

RTS

RETURN

435F 0C

CLASS4

CLC

SHOW ALPHANUMERIC

4360 39

RTS

RETURN

\*

\* GET NEXT CHARACTER FROM BUFFER



\*

4361	FF	40	54	NXTCH	STX	INDEX	SAVE INDEX
4364	FE	40	50	NXTCH2	LDX	BUFPNT	GET POINTER
4367	A6	00		NXTCH3	LDA A	0, X	GET THE CHARACTER
4369	81	0D			CMP A	#CR	IS IT RETURN?
436B	27	0C			BEQ	NXTCH4	
436D	08				INX		BUMP THE POINTER
436E	FF	40	50		STX	BUFPNT	SAVE NEW POSITION
4371	81	20			CMP A	#SPC	CHECK FOR SPACE
4373	26	04			BNE	NXTCH4	
4375	A1	00			CMP A	0, X	NEXT CHAR SPACE?
4377	27	EE			BEQ	NXTCH3	SKIP IF SO
4379	FE	40	54	NXTCH4	LDX	INDEX	RESORE INDEX
437C	20	C4			BRA	CLASS	GO CLASSIFY

\*

\* SKIP ALL SPACES

\*

437E	FF	40	52	SKPSPC	STX	CRSAVE	SAVE INDEX
4381	FE	40	50		LDX	BUFPNT	GET POINTER
4384	A6	00		SKPSP2	LDA A	0, X	GET CHARACTER
4386	81	20			CMP A	#SPC	IS IT SPACE?
4388	26	03			BNE	SKPSP4	
438A	08				INX		BUMP TO NEXT
438B	20	F7			BRA	SKPSP2	REPEAT
438D	FF	40	50	SKPSP4	STX	BUFPNT	SET POINTER
4390	FE	40	52		LDX	CRSAVE	RESTORE INDEX
4393	39				RTS		

\*

\* PREVIEW NEXT CHARACTER

\*

4394	FF	40	52	PRVW	STX	CRSAVE	SAVE X
4397	FE	40	50		LDX	BUFPNT	POINT TO CHARACTER
439A	A6	00			LDA A	0, X	GET CHARACTER
439C	FE	40	52		LDX	CRSAVE	RESET X
439F	39				RTS		RETURN

\*

\* WORK IN HEX VALUE

\*

43A0	37			WRKHx	PSH B		SAVE IND
43A1	C6	04			LDA B	#4	SET UP COUNT
43A3	78	40	5E	WRKHx4	ASL	VALUE+1	SHIFT OVER 4
43A6	79	40	5D		ROL	VALUE	
43A9	5A				DEC B		DEC THE COUNT
43AA	26	F7			BNE	WRKHx4	LOOP TIL DONE
43AC	33				PUL B		RESTORE IND
43AD	BB	40	5E		ADD A	VALUE+1	
43B0	B7	40	5E		STA A	VALUE+1	SET NEW DIGIT



```

43B3 5C          INC B          SET INDICATOR
43B4 39          RTS

*
* GET HEX VALUE FROM BUFFER
*

43B5 5F          GETHEX CLR B          CLEAR INDICATOR
43B6 F7 40 5D    STA B VALUE        CLEAR WORK SPACE
43B9 F7 40 5E    STA B VALUE+1
43BC BD 43 61    GETHE2 JSR NXTCH      GET CHARACTER
43BF 25 0E       BCS GETHE8          GRAPHICS?
43C1 8D 11       BSR TSTHEX         TEST FOR HEX
43C3 25 04       BCS GETHE6          ERROR?
43C5 8D D9       BSR WRKHX
43C7 20 F3       BRA GETHE2
43C9 BD 43 61    GETHE6 JSR NXTCH      GET CHARACTER
43CC 24 FB       BCC GETHE6          WAIT FOR TERM
43CE 39          RTS                ERROR RETURN
43CF FE 40 5D    GETHE8 LDX VALUE      GET VALUE
43D2 0C          CLC                CLEAR ERRORS
43D3 39          RTS                RETURN

*
* TEST FOR HEX DIGIT
*

43D4 81 60       TSTHEX CMP A #$60    IS IT LOWER?
43D6 23 02       BLS TSTHE1
43D8 80 20       SUB A #$20          MAKE UPPER
43DA 80 47       TSTHE1 SUB A #'G     REMOVE BIAS
43DC 2A 0E       BPL TSTHE4
43DE 8B 06       ADD A #6            CHECK RANGE
43E0 2A 04       BPL TSTHE2          ERROR?
43E2 8B 07       ADD A #7            ADD BACK IN
43E4 2A 06       BPL TSTHE4          ERROR?
43E6 8B 0A       TSTHE2 ADD A #10     FINAL BIAS
43E8 2B 02       BMI TSTHE4          ERROR?
43EA 0C          TSTHE3 CLC           CLEAR ERRORS
43EB 39          RTS                RETURN
43EC 0D          TSTHE4 SEC           SET ERROR
43ED 39          RTS                RETURN

*
* INPUT DECIMAL NUMBER
*

43EE 7F 40 5D    INDEC CLR VALUE      CLEAR WORK
43F1 7F 40 5E    CLR VALUE+1
43F4 5F          CLR B
43F5 BD 43 61    INDEC2 JSR NXTCH      GET CHARACTER
43F8 25 D5       BCS GETHE8          TERM?
43FA 81 39       CMP A #'9          CHECK FOR NUMBER
43FC 22 CB       BHI GETHE6

```



43FE 84 0F	AND A	#\$F	MASK NUMBER
4400 37	PSH B		SAVE COUNT
4401 36	PSH A		SAVE NUMBER
4402 B6 40 5D	LDA A	VALUE	GET VALUE
4405 F6 40 5E	LDA B	VALUE+1	
4408 58	ASL B		DO TIMES 6
4409 49	ROL A		
440A 58	ASL B		
440B 49	ROL A		
440C 58	ASL B		
440D 49	ROL A		
440E 78 40 5E	ASL	VALUE+1	TIMES 2
4411 79 40 5D	ROL	VALUE	
4414 FB 40 5E	ADD B	VALUE+1	ADD IN NEW
4417 B9 40 5D	ADC A	VALUE	
441A F7 40 5E	STA B	VALUE+1	SAVE NEW
441D 33	PUL B		GET NUMBER
441E FB 40 5E	ADD B	VALUE+1	
4421 89 00	ADC A	#0	
4423 F7 40 5E	STA B	VALUE+1	SAVE RESULT
4426 B7 40 5D	STA A	VALUE	
4429 33	PUL B		GET COUNT
442A 5C	INC B		BUMP COUNT
442B 20 C8	BRA	INDEC2	REPEAT

\*  
\* ADD B TO X  
\*

442D FF 40 52	ADDBX	STX	CRSAVE	PUT INDEX
4430 FB 40 53		ADD B	CRSAVE+1	ADD IN B
4433 F7 40 53		STA B	CRSAVE+1	SAVE NEW
4436 24 03		BCC	ADDBX2	
4438 7C 40 52		INC	CRSAVE	BUMP MSB
443B FE 40 52	ADDBX2	LDX	CRSAVE	GET NEW VALUE
443E 39		RTS		RETURN

\*  
\* GO SIMULATE PROGRAM  
\*

443F BD 4C 63	GO	JSR	TSTTRM	CHECK TERM
4442 26 32		BNE	SIMUL0	ERROR?
4444 20 4A		BRA	SIMUL2	GO TO IT!

\*  
\* JUMP TO ADDRESS  
\*

4446 BD 4C 63	JUMP	JSR	TSTTRM	CHECK TERM
4449 27 2B		BEQ	SIMUL0	ERROR?
444B 20 3B		BRA	SIMUL5	GO TO IT!

\*



\* LOOK FOR BREAKPOINT

\*

444D	CE	3C	00	LKBP	LDX	#BPTAB	POINT TO TABLE
4450	BC	40	AE	LKBP3	CPX	BPEND	END OF TABLE?
4453	27	13			BEQ	LKBP6	
4455	A1	00			CMP A	0,X	CHECK ADDRESS
4457	26	05			BNE	LKBP4	
4459	E1	01			CMP B	1,X	
445B	26	01			BNE	LKBP4	MATCH?
445D	39				RTS		RETURN - FOUND!
445E	08			LKBP4	INX		BUMP TO NEXT ENTRY
445F	08				INX		
4460	08				INX		
4461	08				INX		
4462	08				INX		
4463	08				INX		
4464	08				INX		
4465	08				INX		
4466	20	E8			BRA	LKBP3	REPEAT
4468	86	01		LKBP6	LDA A	#1	SET NO FIND
446A	39				RTS		RETURN

\*

\* OPERATOR HALT

\*

446B	CE	5A	D8	OHALT	LDX	#OPHST	POINT TO STRING
446E	7E	45	3E		JMP	SIMUL7	GO REPORT

\*

\* SIMULATE PROGRAM

\*

4471	BD	4C	63	SIMUL	JSR	TSTTRM	CHECK TERM
4474	26	03			BNE	SIMUL1	ERROR?
4476	7E	41	CE	SIMUL0	JMP	SYNER	REPORT ERROR
4479	CE	00	00	SIMUL1	LDX	#0	CLEAR REGISTERS
447C	FF	40	A7		STX	STATES	
447F	FF	40	A9		STX	STATES+2	
4482	7F	40	89		CLR	WAITF	
4485	7F	40	99		CLR	NESTC	SET NEST = 0
4488	BD	43	B5	SIMUL15	JSR	GETHEX	GET ADDRESS
448B	25	E9			BCS	SIMUL0	ERROR?
448D	FF	40	93		STX	PC	SET PC
4490	FE	40	93	SIMUL2	LDX	PC	GET PC
4493	FF	40	97		STX	OLDPC	
4496	BC	40	73		CPX	END	FINISHED?
4499	26	03			BNE	SIMUL3	
449B	7E	45	4A		JMP	STOP	DO STOP
449E	FE	41	0F	SIMUL3	LDX	ACIA	POINT TO PORT
44A1	A6	00			LDA A	0,X	GET STATUS
44A3	46				ROR A		
44A4	24	46			BCC	SIMUL5	CHARACTER TYPED?



44A6 A6 01	30 4106	LDA A	1,X	GET CHARACTER
44A8 84 7F	01	AND A	#\$7F	MASK PARITY
44AA 81 03		CMP A	#\$3	IS IT 1C?
44AC 27 BD		BEQ	OHALT	
44AE B1 41 14		CMP A	ESC	IS IT ESCAPE?
44B1 26 05		BNE	SIMU35	
44B3 B7 40 8A		STA A	PAUSF	SET FLAG
44B6 20 34		BRA	SIMUL5	
44B8 81 13	SIMU35	CMP A	#\$13	IS IT 1S?
44BA 26 0C		BNE	SIMUL4	
44BC B6 40 8B		LDA A	DELAY	GET DELAY
44BF 81 FF		CMP A	#\$FF	IS IT SLOWEST?
44C1 27 29		BEQ	SIMUL5	
44C3 7C 40 8B		INC	DELAY	SLOW DOWN
44C6 20 24		BRA	SIMUL5	
44C8 81 06	SIMUL4	CMP A	#\$06	IS IT 1F?
44CA 26 0A		BNE	SIMU45	
44CC B6 40 8B		LDA A	DELAY	GET DELAY
44CF 27 1B		BEQ	SIMUL5	FASTEST?
44D1 7A 40 8B		DEC	DELAY	SPEED UP
44D4 20 16		BRA	SIMUL5	
44D6 81 0E	SIMU45	CMP A	#\$E	IS IT 1N?
44D8 26 08		BNE	SIMU48	
44DA CE 00 01		LDX	#1	SET COUNT TO 1
44DD FF 40 7B		STX	NMIC2	
44E0 20 0A		BRA	SIMUL5	
44E2 81 09	SIMU48	CMP A	#\$09	IS IT 1I?
44E4 26 06		BNE	SIMUL5	
44E6 CE 00 01		LDX	#1	SET COUNT TO 1
44E9 FF 40 7F		STX	IRQC2	
44EC 7D 40 A3	SIMUL5	TST	NOBKF	DO BP?
44EF 26 0E		BNE	SIMUL6	
44F1 B6 40 93		LDA A	PC	GET PC
44F4 F6 40 94		LDA B	PC+1	
44F7 BD 44 4D		JSR	LKBP	CHECK BREAKPOINT
44FA 26 03		BNE	SIMUL6	
44FC 7E 49 54		JMP	PRBP	DO BREAK POINT
44FF B6 40 8B	SIMUL6	LDA A	DELAY	GET DELAY
4502 27 09		BEQ	SIMU65	IS THERE ANY?
4504 CE 01 80	SIMU61	LDX	#\$0180	SET COUNTER
4507 09	SIMU62	DEX		DEC THE COUNT
4508 26 FD		BNE	SIMU62	DELAY HERE
450A 4A		DEC A		FINISHED?
450B 26 F7		BNE	SIMU61	REPEAT
450D BD 45 8E	SIMU65	JSR	TRCOL	
4510 FE 40 93		LDX	PC	ADJUST PC
4513 FF 40 97		STX	OLDPC	
4516 BD 48 C7		JSR	PRREG	
4519 FE 40 87		LDX	SIMCNT	GET COUNT
451C 27 06		BEQ	SIMU67	
451E 09		DEX		DEC BY ONE
451F FF 40 87		STX	SIMCNT	SAVE NEW
4522 27 23		BEQ	EXIT	FINISHED?
4524 7D 40 A4	SIMU67	TST	CFLG	COUNT ENABLED?



4527	26	03		BNE	SIMU69	
4529	7E	44	90	SIMU68	JMP	SIMUL2
452C	FE	40	75	SIMU69	LDX	COUNT
452F	09			DEX		DEC COUNT
4530	FF	40	75	STX	COUNT	
4533	26	F4		BNE	SIMU68	
4535	FE	40	77	LDX	MAXC	GET VALUE
4538	FF	40	75	STX	COUNT	RESET COUNT
453B	CE	59	B4	LDX	#CTST	POINT TO STRING
453E	BD	42	85	SIMUL7	JSR	PSTRNG
4541	CE	40	93	LDX	#PC	PRINT IT
4544	BD	43	26	JSR	OUTADR	PRINT ADDRESS
4547	7E	41	95	EXIT	JMP	EXEC
454A	CE	59	A9	STOP	LDX	#STPST
454D	20	EF		BRA	SIMUL7	POINT TO STRING

\*  
\* IO CHECK  
\*

454F	B6	40	9C	DOIOI	LDA A	PRON	TEST IF TRACE ON
4552	27	1D			BEQ	DOIO	
4554	B1	40	99		CMP A	NESTC	CHECK NEST
4557	23	18			BLS	DOIO	
4559	CE	59	88		LDX	#INPST	POINT TO STRING
455C	20	0D			BRA	DOIO02	
455E	B6	40	9C	DOIO0	LDA A	PRON	TRACE ON?
4561	27	0E			BEQ	DOIO	
4563	B1	40	99		CMP A	NESTC	CHECK NEST
4566	23	09			BLS	DOIO	
4568	CE	59	8D		LDX	#OUTST	POINT TO STRING
456B	BD	42	87	DOIO02	JSR	PDATA1	PRINT STRING
456E	FE	40	93		LDX	PC	GET PC
4571	FF	40	D5	DOIO	STX	OP+1	SAVE AS JMP ADDR
4574	86	BD			LDA A	#\$BD	SETUP JSR
4576	B7	40	D4		STA A	OP	
4579	BD	47	02		JSR	TRCOL5	GO DO IT
457C	86	39			LDA A	#\$39	SETUP RTS
457E	B7	40	D4		STA A	OP	
4581	C6	01			LDA B	#1	
4583	F7	40	9A		STA B	OPCNT	
4586	7E	47	B0		JMP	PSUBR	DO RETURN
4589	CE	5A	CC	DOMON	LDX	#HMONS	POINT TO STRING
458C	20	B0			BRA	SIMUL7	REPORT XFR

\*  
\* TRACE ONE LINE  
\*

458E	86	01		TRCOL	LDA A	#\$01	NOP CODE
4590	B7	40	D5		STA A	OP+1	
4593	B7	40	D6		STA A	OP+2	
4596	7F	40	A2		CLR	EAFL	CLEAR FLAG
4599	FE	40	93		LDX	PC	



459C BC 41 07		CPX	INV	
459F 27 AE		BEQ	DOIOI	
45A1 BC 41 0A		CPX	OUTV	
45A4 27 B8		BEQ	DOIOO	
45A6 BC 41 0D		CPX	MONV	MONITOR REFERENCE?
45A9 27 DE		BEQ	DOMON	
45AB FE 40 C6		LDX	SMEND	SET UP POINTERS
45AE FF 40 CE		STX	TABE	
45B1 CE 3F 00		LDX	#SMTAB	
45B4 B6 40 93		LDA A	PC	GET PC
45B7 F6 40 94		LDA B	PC+1	
45BA BD 4B 77		JSR	CHECK	CHECK SIM PROT
45BD 07		TPA		
45BE FE 40 93		LDX	PC	RESET PC
45C1 06		TAP		
45C2 27 AD		BEQ	DOIO	DO IN REAL TIME
45C4 A6 00		LDA A	0, X	GET OPCODE
45C6 B7 40 D4		STA A	OP	
45C9 BD 48 85		JSR	FNDOP	FIND OPCODE
45CC 7D 40 9B	TRCOL1	TST	MODE	CHECK MODE
45CF 26 03		BNE	TRC011	
45D1 7E 46 AC		JMP	TRC015	
45D4 B6 40 B8	TRC011	LDA A	INCOD	GET CODE
45D7 27 23		BEQ	TRCHK2	ILLEGAL?
45D9 84 0F		AND A	##F	MASK BITS
45DB 81 07		CMP A	#7	IS IT RTS?
45DD 26 08		BNE	TRCHK	
45DF 7D 40 99		TST	NESTC	CHECK NEST
45E2 26 03		BNE	TRCHK	
45E4 7E 4E 56		JMP	NSTER	REPORT NEST ERROR
45E7 7D 40 A6	TRCHK	TST	XFRT	XFR TRAP ON?
45EA 27 13		BEQ	TRCHK4	
45EC 7D 40 A3		TST	NOBKf	NO BREAK?
45EF 26 0E		BNE	TRCHK4	
45F1 81 03		CMP A	#3	IS IT BRA?
45F3 27 04		BEQ	TRCHK1	
45F5 81 05		CMP A	#5	IS IT JMP?
45F7 26 06		BNE	TRCHK4	
45F9 7E 4E 51	TRCHK1	JMP	XFRTRP	REPORT XFR TRAP
45FC 7E 48 7F	TRCHK2	JMP	ILOPTR	REPORT ILL OPCODE
45FF 81 04	TRCHK4	CMP A	#4	IS IT BSR?
4601 27 04		BEQ	TRCHK5	
4603 81 06		CMP A	#6	IS IT JSR?
4605 26 0E		BNE	TRCHK6	
4607 B6 40 A5	TRCHK5	LDA A	NSTRP	CHECK NEST TRAP
460A 27 09		BEQ	TRCHK6	
460C 4A		DEC A		DEC THE COUNT
460D B1 40 99		CMP A	NESTC	CHECK CURRENT NEST
4610 22 03		BHI	TRCHK6	
4612 7E 4E 3B		JMP	TRNEST	REPORT TRAP
4615 A6 03	TRCHK6	LDA A	3, X	GET CODE
4617 84 0F		AND A	##F	MASK STATE COUNT
4619 BB 40 AA		ADD A	STATES+3	ADD IN STATES
461C 19		DAA		



461D B7 40 AA		STA A	STATES+3	
4620 24 1F		BCC	TRCHK7	
4622 86 01		LDA A	#1	SET UP A ONE
4624 BB 40 A9		ADD A	STATES+2	
4627 19		DAA		
4628 B7 40 A9		STA A	STATES+2	
462B 24 14		BCC	TRCHK7	
462D 86 01		LDA A	#1	
462F BB 40 A8		ADD A	STATES+1	
4632 19		DAA		
4633 B7 40 A8		STA A	STATES+1	
4636 24 09		BCC	TRCHK7	
4638 86 01		LDA A	#1	
463A BB 40 A7		ADD A	STATES	
463D 19		DAA		
463E B7 40 A7		STA A	STATES	
4641 7D 40 AB	TRCHK7	TST	ITRF	ITRAP ON?
4644 27 11		BEQ	TRC012	
4646 B6 40 D4		LDA A	OP	GET OP CODE
4649 81 3B		CMP A	##3B	CHECK TYPE
464B 25 0A		BLO	TRC012	
464D 81 3F		CMP A	##3F	
464F 22 06		BHI	TRC012	
4651 CE 59 D6		LDX	#ITRST	POINT TO STRING
4654 7E 45 3E		JMP	SIMUL7	REPORT TRAP
4657 BD 4B 12	TRC012	JSR	PRCH	CHECK PROTECTION
465A B6 40 D4		LDA A	OP	GET OP CODE
465D 81 37		CMP A	##37	CHECK FOR PSH OR PUL
465F 22 15		BHI	TRC125	
4661 81 32		CMP A	##32	
4663 25 11		BLO	TRC125	
4665 81 33		CMP A	##33	IS IT PUL?
4667 22 04		BHI	TRCSP1	
4669 C6 01		LDA B	#1	SET OFFSET
466B 20 06		BRA	TRCSP2	
466D 81 36	TRCSP1	CMP A	##36	IS IT PSH?
466F 25 05		BLO	TRC125	
4671 C6 FF		LDA B	#-1	SET OFFSET
4673 BD 56 C3	TRCSP2	JSR	CHKSP	CHECK SP
4676 B6 40 93	TRC125	LDA A	PC	GET PC
4679 F6 40 94		LDA B	PC+1	
467C FE 40 C0		LDX	NXTPC	POINT TO TABLE
467F A7 00		STA A	0, X	MAKE ENTRY
4681 E7 01		STA B	1, X	
4683 08		INX		BUMP TO NEXT
4684 08		INX		
4685 8C 3F 00		CPX	#ENDPC	END OF TABLE?
4688 26 03		BNE	TRC013	
468A CE 3D 00		LDX	#STRTPC	RESET POINTER
468D FF 40 C0	TRC013	STX	NXTPC	SAVE POSITION
4690 FE 40 7B		LDX	NMIC2	CHECK INTERRUPTS
4693 27 09		BEQ	TRC014	
4695 09		DEX		
4696 FF 40 7B		STX	NMIC2	NMI?



4699 26 03		BNE	TRC014	
469B 7E 48 3A		JMP	PNMI	GO DO NMI
469E FE 40 7F	TRC014	LDX	IRQC2	CHECK IRQ
46A1 27 09		BEQ	TRC015	
46A3 09		DEX		
46A4 FF 40 7F		STX	IRQC2	
46A7 26 03		BNE	TRC015	
46A9 7E 48 1D		JMP	PIRQ	DO IRQ
46AC 7D 40 89	TRC015	TST	WAITF	WAITING FOR INT?
46AF 27 01		BEQ	TRCOL2	
46B1 39		RTS		RETURN
46B2 FE 40 93	TRCOL2	LDX	PC	GET PC
46B5 08		INX		BUMP TO NEXT
46B6 7F 40 A3		CLR	NOBKF	
46B9 F6 40 9A		LDA B	OPCNT	GET COUNT
46BC C1 02		CMP B	#2	CHECK OP LENGTH
46BE 25 15		BLO	OP1LNG	
46C0 22 07		BHI	OP3LNG	
46C2 A6 00	OP2LNG	LDA A	0, X	
46C4 B7 40 D5		STA A	OP+1	
46C7 20 0B		BRA	TRCOL3	
46C9 A6 00	OP3LNG	LDA A	0, X	
46CB B7 40 D5		STA A	OP+1	
46CE 08		INX		
46CF A6 00		LDA A	0, X	
46D1 B7 40 D6		STA A	OP+2	
46D4 08	TRCOL3	INX		BUMP PC
46D5 FF 40 93	OP1LNG	STX	PC	SAVE NEW VALUE
46D8 F6 40 D4		LDA B	OP	
46DB B6 40 B8		LDA A	INCOD	GET CODE
46DE 84 0F		AND A	#\$F	GET TYPE
46E0 81 03		CMP A	#3	IS IT REGULAR?
46E2 25 1E		BLO	TRCOL5	
46E4 26 0C		BNE	TRCOL4	BRANCH TYPE?
46E6 09		DEX		SET XFR ADDR
46E7 09		DEX		
46E8 FF 40 BE		STX	LASTJ	
46EB C1 20		CMP B	##20	IS IT BRA?
46ED 27 50		BEQ	PBRA	
46EF 7E 47 58		JMP	PBXX	CONDITIONAL BRANCH
46F2 CE 58 EB	TRCOL4	LDX	#FUNTAB	POINT TO TABLE
46F5 80 04		SUB A	#4	REMOVE BIAS
46F7 27 05		BEQ	TRC047	
46F9 08	TRC045	INX		BUMP TO NEXT
46FA 08		INX		
46FB 4A		DEC A		DEC THE OFFSET
46FC 26 FB		BNE	TRC045	
46FE EE 00	TRC047	LDX	0, X	GET JUMP VECTOR
4700 6E 00		JMP	0, X	JUMP TO IT
4702 BF 40 95	TRCOL5	STS	SSP	
4705 BE 40 91		LDS	SP	GET REAL STACK
4708 FE 40 8F		LDX	XR	GET X REG
470B F6 40 8E		LDA B	BR	GET B REG
470E B6 40 8D		LDA A	AR	GET A



4711 36		PSH A		
4712 B6 40 8C		LDA A	CC	GET C CODES
4715 06		TAP		
4716 32		PUL A		
4717 7E 40 D4		JMP	OP	DO OPCODE
471A 36	TRCRET	PSH A		
471B 07		TPA		
471C B7 40 8C		STA A	CC	SAVE CC
471F 32		PUL A		
4720 B7 40 8D		STA A	AR	SAVE A
4723 F7 40 8E		STA B	BR	SAVE B
4726 FF 40 8F		STX	XR	SAVE X
4729 BF 40 91		STS	SP	SAVE SP
472C BE 40 95		LDS	SSP	
472F 39		RTS		RETURN

\*  
\* PROCESS BSR  
\*

4730 C6 FE	PBSR	LDA B	#-2	SET OFFSET
4732 BD 56 C3		JSR	CHKSP	CHECK SP CHECK STACK PROT
4735 FE 40 93		LDX	PC	FIX LAST XFR
4738 09		DEX		
4739 09		DEX		
473A FF 40 BE		STX	LASTJ	SET ADDRESS
473D 8D 26		BSR	PSUBC	

\*  
\* PROCESS BRA  
\*

473F 5F	PBRA	CLR B		
4740 B6 40 D5		LDA A	OP+1	CALCULATE ADDRESS
4743 BB 40 94		ADD A	PC+1	
4746 F9 40 93		ADC B	PC	
4749 B7 40 94		STA A	PC+1	SAVE NEW ADDRESS
474C F7 40 93		STA B	PC	
474F 7D 40 D5		TST	OP+1	
4752 2A 03		BPL	PBRA4	
4754 7A 40 93		DEC	PC	
4757 39	PBRA4	RTS		RETURN

\*  
\* PROCESS CONDITIONAL BRANCH  
\*

4758 B6 40 D4	PBXX	LDA A	OP	GET TYPE
475B B7 40 DA		STA A	BXOP	SAVE IT
475E B6 40 8C		LDA A	CC	
4761 06		TAP		
4762 7E 40 DA		JMP	BXOP	DO CON BRANCH

\*



## \* PROCESS SUBROUTINE CALL

\*

4765 BF 40 95	PSUBC	STS	SSP	SAVE STACK
4768 BE 40 91		LDS	SP	
476B B6 40 94		LDA A	PC+1	PUSH ADDRESS
476E 36		PSH A		
476F B6 40 93		LDA A	PC	
4772 36		PSH A		
4773 BF 40 91		STS	SP	SAVE STACK
4776 BE 40 95		LDS	SSP	
4779 7C 40 99		INC	NESTC	SET NEST COUNT
477C 39	PSUBC2	RTS		RETURN

\*

## \* PROCESS JSR

\*

477D 37	PJSR	PSH B		SAVE OPCODE
477E C6 FE		LDA B	#-2	SET OFFSET
4780 BD 56 C3		JSR	CHKSP	CHECK SP
4783 33		PUL B		RESTORE B
4784 C1 AD		CMP B	#\$AD	IS IT INDEXED?
4786 27 16		BEQ	PJSRX	
4788 8D DB		BSR	PSUBC	DO CALL

\*

## \* PROCESS JMP

\*

478A C1 6E	PJMP	CMP B	#\$6E	IS IT INDEXED?
478C 27 12		BEQ	PJMPX	
478E FE 40 93		LDX	PC	GET PC
4791 09		DEX		
4792 09		DEX		
4793 09		DEX		
4794 FF 40 BE		STX	LASTJ	SET LAST XFR
4797 FE 40 D5		LDX	OP+1	
479A FF 40 93		STX	PC	
479D 39		RTS		RETURN

\*

## \* PROCESS INDEXED JSR

\*

479E 8D C5	PJSRX	BSR	PSUBC	
47A0 FE 40 93	PJMPX	LDX	PC	GET PC
47A3 09		DEX		FIX POS
47A4 09		DEX		
47A5 FF 40 BE		STX	LASTJ	SET LAST XFR
47A8 FE 40 8F		LDX	XR	
47AB FF 40 93		STX	PC	
47AE 20 8F		BRA	PBRA	



\*  
\* PROCESS RTS INSTRUCTION  
\*

47B0	C6	02	PSUBR	LDA B	#2	SET OFFSET
47B2	BD	56		JSR	CHKSP	CHECK SP
47B5	FE	40		LDX	PC	SET LAST XFR
47B8	09			DEX		
47B9	FF	40		STX	LASTJ	SAVE ADDRESS
47BC	BF	40		STS	SSP	
47BF	BE	40		LDS	SP	
47C2	32			PUL A		RESET PC
47C3	B7	40		STA A	PC	
47C6	32			PUL A		
47C7	B7	40		STA A	PC+1	
47CA	BF	40		STS	SP	SAVE STACK
47CD	BE	40		LDS	SSP	
47D0	7A	40		DEC	NESTC	FIX NEST COUNT
47D3	39			RTS		

\*  
\* PUSH REGISTERS ON STACK  
\*

47D4	77	40	89	PONS	ASR	WAITF	TEST WAIT STATUS
47D7	25	2D			BCS	PONS2	
47D9	C6	F9			LDA B	#-7	SET OFFSET
47DB	BD	56	C3		JSR	CHKSP	CHECK SP
47DE	BF	40	95		STS	SSP	SAVE STACK
47E1	BE	40	91		LDS	SP	GET MACHINES
47E4	B6	40	94		LDA A	PC+1	PUSH PC
47E7	36				PSH A		
47E8	B6	40	93		LDA A	PC	
47EB	36				PSH A		
47EC	B6	40	90		LDA A	XR+1	PUSH X
47EF	36				PSH A		
47F0	B6	40	8F		LDA A	XR	
47F3	36				PSH A		
47F4	B6	40	8D		LDA A	AR	PUSH A
47F7	36				PSH A		
47F8	B6	40	8E		LDA A	BR	PUSH B
47FB	36				PSH A		
47FC	B6	40	8C		LDA A	CC	PUSH CC
47FF	36				PSH A		
4800	BF	40	91		STS	SP	SAVE STACK
4803	BE	40	95		LDS	SSP	RESTORE
4806	39			PONS2	RTS		RETURN

\*  
\* SET INTERRUPT MASK  
\*

4807	86	10	SEIM	LDA A	##10	SET MASK
------	----	----	------	-------	------	----------



4809 BA 40 8C	ORA A	CC	OR IT IN
480C B7 40 8C	STA A	CC	SAVE NEW
480F 39	RTS		RETURN

\*  
\* PROCESS SWI INSTRUCTION

4810 8D C2	PSWI	BSR	PONS	PUSH ALL
4812 8D F3		BSR	SEIM	SET MASK
4814 FE 41 15		LDX	SWIV	GET VECTOR
4817 EE 00		LDX	0, X	GET INDIRECT
4819 FF 40 93	PSWI2	STX	PC	SET IN PC
481C 39	PSWI4	RTS		RETURN

\*  
\* PROCESS IRQ

481D FE 40 7D	PIRQ	LDX	IRQC	RESET COUNTER
4820 FF 40 7F		STX	IRQC2	
4823 86 10		LDA A	##10	TEST MASK
4825 B5 40 8C		BIT A	CC	
4828 27 03		BEQ	PIRQ1	
482A 7E 46 AC		JMP	TRC015	GO DO NEXT
482D 8D A5	PIRQ1	BSR	PONS	ALL ON STACK
482F 8D D6	PIRQ2	BSR	SEIM	SET MASK
4831 FE 41 17		LDX	IRQV	GET VECTOR
4834 EE 00	PIRQ4	LDX	0, X	GET INDIRECT
4836 FF 40 93		STX	PC	SET NEW PC
4839 39		RTS		RETURN

\*  
\* PROCESS NMI

483A FE 40 79	PNMI	LDX	NMIC	RESET COUNT
483D FF 40 7B		STX	NMIC2	
4840 8D 92		BSR	PONS	ALL ON STACK
4842 8D C3	PNMI2	BSR	SEIM	SET MASK
4844 FE 41 19		LDX	NMIV	GET VECTOR
4847 20 EB		BRA	PIRQ4	

\*  
\* PROCESS WAI INSTRUCTION

4849 8D 89	PWAI	BSR	PONS	PUSH ALL
484B 86 01	PWAI2	LDA A	#1	SET WAIT FLAG
484D B7 40 89		STA A	WAITF	
4850 39		RTS		RETURN

\*  
\* PROCESS RTI INSTRUCTION



\*

4851	C6 07	PRTI	LDA B	#7	SET OFFSET
4853	BD 56 C3		JSR	CHKSP	CHECK SP
4856	BF 40 95		STS	SSP	SAVE STACK
4859	BE 40 91		LDS	SP	GET MACHINE STACK
485C	32		PUL A		PULL CC
485D	B7 40 8C		STA A	CC	
4860	32		PUL A		PULL B
4861	B7 40 8E		STA A	BR	
4864	32		PUL A		PULL A
4865	B7 40 8D		STA A	AR	
4868	32		PUL A		PULL X
4869	B7 40 8F		STA A	XR	
486C	32		PUL A		
486D	B7 40 90		STA A	XR+1	
4870	32		PUL A		PULL PC
4871	B7 40 93		STA A	PC	
4874	32		PUL A		
4875	B7 40 94		STA A	PC+1	
4878	BF 40 91		STS	SP	SAVE STACK
487B	BE 40 95		LDS	SSP	RESTORE
487E	39		RTS		

487F	CE 59 C3	ILOPTR	LDX	#ILOPST	POINT TO STRING
4882	7E 45 3E		JMP	SIMUL7	GO PRINT IT

\*

\* FIND OPCODE IN TABLE

\*

4885	5F	FNDOP	CLR B		
4886	B7 40 C5		STA A	OPPNT2+1	SET POINTER 2
4889	48		ASL A		CALCULATE POSITION
488A	59		ROL B		
488B	48		ASL A		
488C	59		ROL B		
488D	BB 40 D3		ADD A	POPTAB+1	SET POSITION
4890	F9 40 D2		ADC B	POPTAB	
4893	B7 40 C3		STA A	OPPNT+1	SAVE POSITION
4896	F7 40 C2		STA B	OPPNT	
4899	FE 40 C4		LDX	OPPNT2	GET POINTER
489C	A6 00		LDA A	0, X	GET CODE
489E	B7 40 B8		STA A	INCOD	SAVE CODE
48A1	FE 40 C2		LDX	OPPNT	GET POINTER
48A4	5F		CLR B		
48A5	A6 03		LDA A	3, X	GET CODE BYTE
48A7	84 30		AND A	##30	MASK LENGTH
48A9	27 0C		BEQ	FNDOP4	
48AB	81 30		CMP A	##30	IS IT 3?
48AD	27 06		BEQ	FNDOP2	
48AF	81 20		CMP A	##20	IS IT 2?
48B1	27 03		BEQ	FNDOP3	
48B3	20 02		BRA	FNDOP4	



48B5 5C	FNDOP2	INC B	SET LENGTH COUNT
48B6 5C	FNDOP3	INC B	
48B7 5C	FNDOP4	INC B	
48B8 F7 40 9A	FNDOP5	STA B OPCNT	SAVE COUNT
48BB 39	FNDOP6	RTS	RETURN

\*  
\* DUMP REGISTERS COMMAND  
\*

48BC FE 40 93	DREG	LDX	PC	UPDATE OLD PC
48BF FF 40 97		STX	OLDPC	
48C2 8D 0D		BSR	PRREG2	PRINT REGS
48C4 7E 41 95		JMP	EXEC	RETURN TO EXEC

\*  
\* PRINT REGISTERS  
\*

48C7 B6 40 9C	PRREG	LDA A	PRON	CHECK TRACE ON
48CA 27 EF		BEQ	FNDOP6	
48CC B1 40 99		CMP A	NESTC	CHECK NEST LEVEL
48CF 23 EA		BLS	FNDOP6	
48D1 BD 42 94	PRREG2	JSR	PCRLF	PRINT CR & LF
48D4 CE 59 41		LDX	#CCST	PRINT C CODES
48D7 8D 34		BSR	PRRE22	
48D9 CE 40 8C		LDX	#CC	
48DC BD 43 29		JSR	OUTHEX	
48DF CE 59 44		LDX	#AST	PRINT A
48E2 8D 29		BSR	PRRE22	
48E4 CE 40 8D		LDX	#AR	
48E7 BD 43 29		JSR	OUTHEX	
48EA CE 59 48		LDX	#BST	PRINT B
48ED 8D 1E		BSR	PRRE22	
48EF CE 40 8E		LDX	#BR	
48F2 BD 43 29		JSR	OUTHEX	
48F5 CE 59 4C		LDX	#XST	PRINT X
48F8 8D 13		BSR	PRRE22	
48FA CE 40 8F		LDX	#XR	
48FD BD 43 26		JSR	OUTADR	
4900 CE 59 50		LDX	#SPST	PRINT SP
4903 8D 08		BSR	PRRE22	
4905 CE 40 91		LDX	#SP	
4908 BD 43 26		JSR	OUTADR	
490B 20 03		BRA	PRRE25	
490D 7E 42 87	PRRE22	JMP	PDATA1	PRINT STRING
4910 CE 59 54	PRRE25	LDX	#PCST	PRINT PC
4913 8D F8		BSR	PRRE22	
4915 CE 40 93		LDX	#PC	
4918 BD 43 26		JSR	OUTADR	
491B CE 59 58		LDX	#NST	PRINT N
491E 8D ED		BSR	PRRE22	
4920 CE 40 99		LDX	#NESTC	
4923 BD 43 29		JSR	OUTHEX	



4926	7D	40	A1	TST	VFLG	PRINT INDIRECTION?
4929	27	0B		BEQ	PRREG3	
492B	CE	59	5C	LDX	#XVST	POINT TO STRING
492E	8D	DD		BSR	PRRE22	PRINT IT
4930	FE	40	8F	LDX	XR	GET ADDRESS
4933	BD	43	29	JSR	OUTHEX	PRINT CONTENTS
4936	7D	40	A0	PRREG3 TST	FLAGB	CHECK FLAG OPT
4939	27	0B		BEQ	PRREG4	
493B	CE	59	60	LDX	#FMST	POINT TO STRING
493E	8D	CD		BSR	PRRE22	PRINT IT
4940	FE	40	9E	LDX	FLAGA	POINT TO MEMORY
4943	BD	43	26	JSR	OUTADR	PRINT IT
4946	8D	07		PRREG4 BSR	OUTSP	
4948	8D	05		BSR	OUTSP	
494A	7E	4A	43	JMP	DISO	

\*

\* OUTPUT SPACES

\*

494D	8D	00		OUT2SP BSR	OUTSP	OUTPUT SPACE
494F	86	20		OUTSP LDA A	#\$20	
4951	7E	41	09	JMP	PUTCHR	

\*

\* PROCESS BREAK POINT

\*

4954	FF	40	AC	PRBP	STX	BPP05	SAVE BP POSITION
4957	6D	02			TST	2, X	CHECK COUNT
4959	2A	1B			BPL	PRBP2	FORWARDS?
495B	6D	03			TST	3, X	COUNT LEFT?
495D	26	0B			BNE	PRBP1	
495F	A6	02			LDA A	2, X	GET MSB OF COUNT
4961	84	7F			AND A	#\$7F	MASK DIR BIT
4963	26	05			BNE	PRBP1	IS IT ZERO?
4965	BD	51	4D		JSR	RMVBP	REMOVE BREAK POINT
4968	20	1D			BRA	PRBP3	
496A	A6	03		PRBP1	LDA A	3, X	GET COUNT
496C	80	01			SUB A	#1	DEC BY 1
496E	A7	03			STA A	3, X	
4970	24	18			BCC	PRBP4	
4972	6A	02			DEC	2, X	FIX MSB OF COUNT
4974	20	14			BRA	PRBP4	
4976	EE	02		PRBP2	LDX	2, X	GET COUNT
4978	27	10			BEQ	PRBP4	
497A	FE	40	AC		LDX	BPP05	RESTORE POSITION
497D	A6	03			LDA A	3, X	GET COUNT
497F	80	01			SUB A	#1	DEC BY 1
4981	A7	03			STA A	3, X	
4983	24	02			BCC	PRBP3	
4985	6A	02			DEC	2, X	FIX MSB
4987	7E	44	FF	PRBP3	JMP	SIMUL6	RETURN TO SIM
498A	FE	40	AC	PRBP4	LDX	BPP05	RESET POINTER



498D A6 04		LDA A	4, X	GET CONDITIONAL
498F 16		TAB		
4990 27 4D		BEQ	PRB57	NO CONDITION?
4992 7F 40 B9		CLR	EQUALS	
4995 84 03		AND A	##3	MASK EQUALITY
4997 81 02		CMP A	#2	IS IT !=
4999 26 03		BNE	PRB45	
499B 73 40 B9		COM	EQUALS	SET FOR NOT-EQUAL
499E 5D	PRB45	TST B		CHECK MEM REF
499F 2A 06		BPL	PRB46	
49A1 EE 05		LDX	5, X	GET ADDRESS
49A3 6D 00		TST	0, X	CHECK IF ZERO
49A5 20 29		BRA	PRB495	
49A7 CE 58 F9	PRB46	LDX	#RGTAB	POINT TO TABLE
49AA 57		ASR B		ADJUST REG SPEC
49AB 57		ASR B		
49AC 57		ASR B		
49AD 57		ASR B		
49AE 37		PSH B		SAVE REG SPEC
49AF 5A		DEC B		
49B0 27 05		BEQ	PRB475	
49B2 08	PRB47	INX		FIND IN TABLE
49B3 08		INX		
49B4 5A		DEC B		DEC THE COUNT
49B5 26 FB		BNE	PRB47	
49B7 EE 00	PRB475	LDX	0, X	GET REG POSITION
49B9 33		PUL B		RESTORE REG SPEC
49BA C1 04		CMP B	#4	SINGLE BYTE REG?
49BC 22 05		BHI	PRB48	
49BE 4F		CLR A		CLEAR MSB
49BF E6 00		LDA B	0, X	GET VALUE
49C1 20 04		BRA	PRB49	
49C3 A6 00	PRB48	LDA A	0, X	GET 2 BYTE VALUE
49C5 E6 01		LDA B	1, X	
49C7 FE 40 AC	PRB49	LDX	BPPOS	RESET POINTER
49CA A1 05		CMP A	5, X	COMPARE VALUES
49CC 26 0C		BNE	PRB55	
49CE E1 06		CMP B	6, X	
49D0 26 08	PRB495	BNE	PRB55	
49D2 7D 40 B9		TST	EQUALS	TEST EQUALITY
49D5 27 08		BEQ	PRB57	
49D7 7E 44 FF	PRBP5	JMP	SIMUL6	NO COMPARE
49DA 7D 40 B9	PRB55	TST	EQUALS	TEST EQUALITY
49DD 27 F8		BEQ	PRBP5	
49DF FE 40 AC	PRB57	LDX	BPPOS	POINT TO BP
49E2 A6 07		LDA A	7, X	GET ACTION CODE
49E4 B7 40 B0		STA A	BPCODE	SAVE IT
49E7 85 02		BIT A	##2	PRINT REGISTERS?
49E9 27 03		BEQ	PRBP6	
49EB BD 48 D1		JSR	PRREG2	PRINT REGISTERS
49EE B6 40 B0	PRBP6	LDA A	BPCODE	GET CODE
49F1 85 04		BIT A	##4	TRACE ON?
49F3 27 08		BEQ	PRB62	
49F5 C6 FF		LDA B	##FF	SET COUNT



49F7 F7 40 9D	STA B	TRCF	
49FA F7 40 9C	STA B	PRON	SET TRACE ON
49FD 85 08	PRB62 BIT A	#\$8	TRACE OFF?
49FF 27 06	BEQ	PRB65	
4A01 7F 40 9C	CLR	PRON	TRACE OFF
4A04 7F 40 9D	CLR	TRCF	
4A07 85 10	PRB65 BIT A	#\$10	RESET STATE COUNTER?
4A09 27 0C	BEQ	PRB68	
4A0B CE 00 00	LDX	#0	SET UP ZERO
4A0E FF 40 A7	STX	STATES	SET STATES
4A11 FF 40 A9	STX	STATES+2	
4A14 FE 40 AC	LDX	BPP05	RESET POINTER
4A17 85 20	PRB68 BIT A	#\$20	HISTOGRAM?
4A19 27 09	BEQ	PRBP7	
4A1B FE 40 AC	LDX	BPP05	RESET POSITION
4A1E 6C 06	INC	6, X	BUMP COUNTER
4A20 26 02	BNE	PRBP7	CARRY?
4A22 6C 05	INC	5, X	
4A24 85 40	PRBP7 BIT A	#\$40	MESSAGE?
4A26 26 0C	BNE	PRB75	
4A28 85 80	BIT A	#\$80	JUMP TYPE?
4A2A 27 0D	BEQ	PRBP9	
4A2C EE 05	LDX	5, X	GET TRANSFER ADDRESS
4A2E FF 40 93	STX	PC	SET NEW PC
4A31 7E 44 90	JMP	SIMUL2	GO TRACE
4A34 EE 05	PRB75 LDX	5, X	GET MSG ADDRESS
4A36 BD 42 85	JSR	PSTRNG	PRINT MESSAGE
4A39 B6 40 B0	PRBP9 LDA A	BPCODE	GET CODE
4A3C 85 01	BIT A	#\$1	CHECK FOR HALT
4A3E 27 97	BEQ	PRBP5	
4A40 7E 41 95	JMP	EXEC	RETURN TO EXEC

\*  
\* DISASSEMBLE ONE LINE  
\*

4A43 FE 40 97	DIS0 LDX	OLDPC	GET PC
4A46 A6 00	LDA A	0, X	GET OPCODE
4A48 BD 48 85	JSR	FNDOP	FIND OPERAND
4A4B CE 40 97	DIS01 LDX	#OLDPC	POINT TO PC
4A4E BD 43 26	JSR	OUTADR	PRINT IT
4A51 BD 49 4F	JSR	OUTSP	PRINT SPACE
4A54 FE 40 C2	LDX	OPPNT	GET POINTER
4A57 C6 03	LDA B	#3	SET COUNT
4A59 A6 00	DIS02 LDA A	0, X	GET CHARACTER
4A5B BD 41 09	JSR	PUTCHR	OUTPUT IT
4A5E 08	INX		BUMP TO NEXT
4A5F 5A	DEC B		
4A60 26 F7	BNE	DIS02	
4A62 BD 49 4F	JSR	OUTSP	PRINT SPACE
4A65 F6 40 B8	LDA B	INC0D	GET CODE
4A68 17	TBA		
4A69 84 0F	AND A	#\$F	MASK CODE
4A6B 27 13	BEQ	DIS04	



4A6D 81 01		CMP A	#1	CHECK FOR A OR B
4A6F 27 08		BEQ	DIS03	
4A71 81 02		CMP A	#2	CHECK FOR B
4A73 26 0B		BNE	DIS04	
4A75 86 42		LDA A	#'B	SETUP 'B'
4A77 20 02		BRA	DIS035	
4A79 86 41	DIS03	LDA A	#'A	SETUP 'A'
4A7B BD 41 09	DIS035	JSR	PUTCHR	PRINT IT
4A7E 20 03		BRA	DIS045	
4A80 BD 49 4F	DIS04	JSR	OUTSP	PRINT SPACE
4A83 BD 49 4F	DIS045	JSR	OUTSP	
4A86 17		TBA		
4A87 84 70		AND A	##70	CHECK ADDRESS MODE
4A89 27 1E		BEQ	PERR	
4A8B 81 10		CMP A	##10	IMMEDIATE?
4A8D 26 22		BNE	DIS05	
4A8F 86 23		LDA A	#'#	OUTPUT '#'
4A91 BD 41 09		JSR	PUTCHR	
4A94 FE 40 97	DIS047	LDX	OLDPC	GET PC
4A97 08		INX		
4A98 86 24		LDA A	#'\$	OUTPUT '\$'
4A9A BD 41 09		JSR	PUTCHR	
4A9D F6 40 9A		LDA B	OPCNT	GET OP LENGTH
4AA0 C1 02		CMP B	#2	IS IT 2?
4AA2 27 08		BEQ	PBYTE	
4AA4 BD 43 26	PADR	JSR	OUTADR	PRINT ADDRESS
4AA7 20 5B	PADR2	BRA	DIS07	
4AA9 FE 40 97	PERR	LDX	OLDPC	POINT TO BYTE
4AAC BD 43 29	PBYTE	JSR	OUTHEX	PRINT BYTE
4AAF 20 53		BRA	DIS07	
4AB1 81 20	DIS05	CMP A	##20	DIRECT?
4AB3 27 DF		BEQ	DIS047	
4AB5 81 30	DIS055	CMP A	##30	INDEXED?
4AB7 26 1A		BNE	DIS06	
4AB9 FE 40 97		LDX	OLDPC	GET PC
4ABC A6 01		LDA A	1,X	GET OFFSET
4ABE B7 40 5E		STA A	VALUE+1	
4AC1 5F		CLR B		CLEAR FLAG
4AC2 F7 40 5D		STA B	VALUE	SET TO ZERO
4AC5 CE 40 5D		LDX	#VALUE	POINT TO IT
4AC8 BD 42 D5		JSR	OUTDEC	PRINT OFFSET
4ACB CE 59 93		LDX	#IXST	POINT TO STRING
4ACE BD 42 87		JSR	PDATA1	PRINT IT
4AD1 20 31		BRA	DIS07	
4AD3 81 40	DIS06	CMP A	##40	EXTENDED?
4AD5 27 BD		BEQ	DIS047	
4AD7 81 60		CMP A	##60	
4AD9 26 29		BNE	DIS07	
4ADB FE 40 97		LDX	OLDPC	GET PC
4ADE 5F		CLR B		
4ADF A6 01		LDA A	1,X	GET OFFSET
4AE1 36		PSH A		SAVE IT
4AE2 08		INX		BUMP TO NEW PC
4AE3 08		INX		



4AE4 FF 40 BA	STX	RELADR	SAVE ADDRESS
4AE7 BB 40 BB	ADD A	RELADR+1	ADD IN OFFSET
4AEA F9 40 BA	ADC B	RELADR	
4AED B7 40 BB	STA A	RELADR+1	SAVE RESULT
4AF0 F7 40 BA	STA B	RELADR	
4AF3 32	PUL A		GET OFFSET
4AF4 4D	TST A		IS IT PLUS?
4AF5 2A 03	BPL	DIS067	
4AF7 7A 40 BA	DEC	RELADR	ADJUST ADDR.
4AFA 86 24	LDA A	#'\$	PRINT '\$'
4AFC BD 41 09	JSR	PUTCHR	
4AFF CE 40 BA	LDX	#RELADR	POINT TO ADDRESS
4B02 20 A0	BRA	PADR	PRINT IT
4B04 F6 40 9A	LDA B	OPCNT	GET LENGTH
4B07 FE 40 97	LDX	OLDPC	
4B0A 08	INX		BUMP PC
4B0B 5A	DEC B		DEC THE COUNT
4B0C 26 FC	BNE	DIS08	
4B0E FF 40 97	STX	OLDPC	FIX PC
4B11 39	RTS		RETURN

\*  
\* CHECK PROTECTION TABLES  
\*

4B12 B6 40 93	PRCH	LDA A	PC	GET PC
4B15 F6 40 94		LDA B	PC+1	
4B18 FE 40 C8		LDX	EXEND	SET TAB END
4B1B FF 40 CE		STX	TABE	
4B1E CE 3F 20		LDX	#EXTAB	POINT TO EX TABLE
4B21 8D 54		BSR	CHECK	GO CHECK ENTRY
4B23 26 05		BNE	PRCH2	FOUND?
4B25 CE 59 FD		LDX	#EXPST	POINT TO STRING
4B28 20 0A		BRA	PRCH3	
4B2A B6 40 93	PRCH2	LDA A	PC	GET PC
4B2D 8D 3F		BSR	MPR	CHECK MEM PROT
4B2F 26 0F		BNE	PRCH4	FOUND?
4B31 CE 5A 15		LDX	#MMP1ST	POINT TO STRING
4B34 BD 42 85	PRCH3	JSR	PSTRNG	PRINT IT
4B37 CE 40 93		LDX	#PC	POINT TO PC
4B3A BD 43 26		JSR	OUTADR	PRINT ADDRESS
4B3D 7E 41 95		JMP	EXEC	RETURN TO EXEC
4B40 BD 4B 9D	PRCH4	JSR	CEA	FIND E. A.
4B43 25 57		BCS	CHECK6	FOUND?
4B45 B6 40 D0		LDA A	EADR	GET ADDRESS
4B48 F6 40 D1		LDA B	EADR+1	
4B4B 7D 40 B8		TST	INCOD	WRITE CODE?
4B4E 2A 12		BPL	PRCH5	
4B50 FE 40 CA		LDX	WPEND	SET TABLE END
4B53 FF 40 CE		STX	TABE	
4B56 CE 3F 40		LDX	#WPTAB	POINT TO WP TABLE
4B59 8D 1C		BSR	CHECK	GO CHECK ENTRY
4B5B 26 05		BNE	PRCH5	FOUND?
4B5D CE 5A 09		LDX	#WPST	POINT TO STRING



```

4B60 20 D2      BRA      PRCH3
4B62 B6 40 D0    PRCH5    LDA A    EADR      GET ADDRESS
4B65 8D 07      BSR      MPR      CHECK MEM PROT
4B67 26 33      BNE      CHECK6    FOUND?
4B69 CE 5A 26    LDX      #MMP2ST  POINT TO STRING
4B6C 20 C6      BRA      PRCH3

```

```

*
* CHECK MEMORY PROTECT TABLE
*

```

```

4B6E FE 40 CC    MPR      LDX      MEND      SET END
4B71 FF 40 CE    STX      TABE
4B74 CE 3F 60    LDX      #MTAB     POINT TO TABLE

```

```

*
* CHECK FOR ADDRESS IN PR TABLE
*

```

```

4B77 BC 40 CE    CHECK    CPX      TABE      END OF TABLE?
4B7A 27 1E      BEQ      CHECK5
4B7C A1 00      CMP A    0, X      CHECK 1ST ENTRY
4B7E 25 14      BLO      CHECK4
4B80 22 06      BHI      CHECK2
4B82 E1 01      CMP B    1, X
4B84 25 0E      BLO      CHECK4
4B86 27 0A      BEQ      CHECK3
4B88 A1 02      CHECK2    CMP A    2, X      CHECK 2ND ENTRY
4B8A 22 08      BHI      CHECK4
4B8C 25 04      BLO      CHECK3
4B8E E1 03      CMP B    3, X
4B90 22 02      BHI      CHECK4
4B92 4F        CHECK3    CLR A      SHOW FOUND
4B93 39        RTS        RETURN
4B94 08        CHECK4    INX      MOVE TO NEXT
4B95 08        INX
4B96 08        INX
4B97 08        INX
4B98 20 DD      BRA      CHECK      REPEAT
4B9A 86 01      CHECK5    LDA A    #1      SHOW NO FIND
4B9C 39        CHECK6    RTS        RETURN

```

```

*
* CALCULATE EFFECTIVE ADDRESS
*

```

```

4B9D B6 40 B8    CEA      LDA A    INCOD     GET CODE
4BA0 84 70      AND A    #70      MASK BITS
4BA2 81 10      CMP A    #10     CHECK TYPES
4BA4 23 3E      BLS      CEA6
4BA6 81 50      CMP A    #50
4BA8 24 3A      BHS      CEA6
4BAA 7F 40 D0    CLR      EADR      ZERO E. A. AREA
4BAD 7F 40 D1    CLR      EADR+1

```



4BB0 B7 40 A2	STA A	EAFL	SET FLAG
4BB3 FE 40 93	LDX	PC	GET PC
4BB6 81 40	CMP A	##40	EXTENDED?
4BB8 27 19	BEQ	CEA2	
4BBA 81 20	CMP A	##20	DIRECT?
4BBC 27 1B	BEQ	CEA3	
4BBE A6 01	LDA A	1,X	INDEXED
4BC0 FE 40 8F	LDX	XR	GET INDEX REG
4BC3 FF 40 D0	STX	EADR	SAVE VALUE
4BC6 BB 40 D1	ADD A	EADR+1	ADD IN OFFSET
4BC9 B7 40 D1	STA A	EADR+1	SAVE RESULT
4BCC 24 14	BCC	CEA5	
4BCE 7C 40 D0	INC	EADR	
4BD1 20 0F	BRA	CEA5	
4BD3 A6 01	CEA2 LDA A	1,X	GET EXT ADDRESS
4BD5 E6 02	LDA B	2,X	
4BD7 20 03	BRA	CEA4	
4BD9 4F	CEA3 CLR A		CLEAR FOR DIRECT
4BDA E6 01	LDA B	1,X	GET ADDRESS
4BDC B7 40 D0	CEA4 STA A	EADR	SET ADDRESS
4BDF F7 40 D1	STA B	EADR+1	
4BE2 0C	CEA5 CLC		SHOW ADDRESS
4BE3 39	RTS		RETURN
4BE4 0D	CEA6 SEC		SHOW NO ADDRESS
4BE5 39	RTS		

\*  
 \* SET SINGLE STEP MODE  
 \*

4BE6 BD 42 94	SSM JSR	PCRLF	PRINT CR & LF
4BE9 CE 00 01	LDX	#1	SET DEFAULT
4BEC FF 40 85	STX	STPCNT	
4BEF BD 4C 63	JSR	TSTTRM	CHECK TERM
4BF2 27 08	BEQ	SSM1	
4BF4 BD 43 EE	JSR	INDEC	GET COUNT
4BF7 25 3B	BCS	EQVAL2	ERROR?
4BF9 FF 40 85	STX	STPCNT	SET COUNT
4BFC FE 40 85	SSM1 LDX	STPCNT	RESET COUNTER
4BFF FF 40 87	STX	SIMCNT	
4C02 BD 45 8E	SSM15 JSR	TRCOL	TRACE ONE LINE
4C05 FE 40 93	LDX	PC	RESET PC
4C08 FF 40 97	STX	OLDPC	
4C0B FE 40 87	LDX	SIMCNT	GET COUNTER
4C0E 09	DEX		DEC BY 1
4C0F FF 40 87	STX	SIMCNT	SAVE NEW
4C12 26 EE	BNE	SSM15	
4C14 BD 48 D1	JSR	PRREG2	PRINT REGS
4C17 BD 41 06	SSM3 JSR	GETCHR	GET CHARACTER
4C1A 81 0D	CMP A	##D	IS IT CR?
4C1C 27 09	BEQ	SSM2	
4C1E 81 20	CMP A	##20	IS IT SPACE?
4C20 26 F5	BNE	SSM3	
4C22 B7 40 A3	STA A	NOBK	SET NO BREAK



4C25 20 D5  
4C27 7E 41 95 SSM2

BRA  
JMP SSM1  
EXEC

RETURN TO EXEC

\*  
\* CHECK EQUALS VALUE  
\*

4C2A 8D 31  
4C2C 26 06  
4C2E BD 43 EE  
4C31 25 01  
4C33 39  
4C34 7E 41 CE

EQVAL BSR  
BNE  
JSR  
BCS  
RTS  
EQVAL2 JMP

TSTEQ  
EQVAL2  
INDEC  
EQVAL2  
SYNER

TEST FOR EQUALS  
ERROR?  
GET COUNT  
ERROR?  
REPORT ERROR

\*  
\* CHECK HEX EQUALS VALUE  
\*

4C37 8D 24  
4C39 26 F9  
4C3B BD 43 B5  
4C3E 25 F4  
4C40 39

HEQVAL BSR  
BNE  
JSR  
BCS  
RTS

TSTEQ  
EQVAL2  
GETHEX  
EQVAL2

CHECK FOR EQUALS  
ERROR?  
GET HEX VALUE  
ERROR?  
RETURN

\*  
\* SET INSTRUCTION COUNT TRAP  
\*

4C41 8D E7  
4C43 7F 40 A4  
4C46 FF 40 75  
4C49 FF 40 77  
4C4C 27 03  
4C4E 7C 40 A4  
4C51 20 D4

INST BSR  
CLR  
STX  
STX  
BEQ  
INC  
INST2 BRA

EQVAL  
CFLG  
COUNT  
MAXC  
INST2  
CFLG  
SSM2

TEST VALUE  
CLEAR FLAG  
SAVE COUNT  
SET VALUE  
IS IT ZERO?  
SET MODE  
RETURN

\*  
\* SET NEST TRAP  
\*

4C53 8D D5  
4C55 B6 40 5E  
4C58 B7 40 A5  
4C5B 20 F4

NEST BSR  
LDA A  
STA A  
BRA

EQVAL  
VALUE+1  
NSTRP  
INST2

TEST VALUE  
GET VALUE  
SAVE COUNT

\*  
\* TEST FOR EQUALS  
\*

4C5D B6 40 68  
4C60 81 3D  
4C62 39

TSTEQ LDA A LSTTRM  
CMP A #'=  
RTS

CHECK TERM  
IS IT EQUALS?

\*



## \* TEST TERMINATOR

\*

4C63 B6 40 68	TSTTRM	LDA A	LSTTRM	GET TERM
4C66 81 0D		CMP A	##D	IS IT CR?
4C68 39		RTS		RETURN

\*

## \* TEST FOR ON OR OFF

\*

4C69 8D F2	TSTON	BSR	TSTEQ	TEST FOR EQUALS
4C6B 26 C7		BNE	EQVAL2	ERROR?
4C6D BD 43 61		JSR	NXTCH	GET CHARACTER
4C70 84 5F		AND A	##5F	MAKE UPPER
4C72 81 4F		CMP A	#'0	IS IT '0'?
4C74 26 BE		BNE	EQVAL2	ERROR?
4C76 BD 43 61		JSR	NXTCH	GET CHARACTER
4C79 84 5F		AND A	##5F	
4C7B 81 4E		CMP A	#'N	IS IT 'N'?
4C7D 39		RTS		RETURN

\*

## \* PRINT TRAIL REGISTER

\*

4C7E CE 59 E1	TRAIL	LDX	#TRST	POINT TO STRING
4C81 BD 42 85		JSR	PSTRNG	PRINT IT
4C84 CE 40 BE		LDX	#LASTJ	GET VALUE
4C87 BD 43 26		JSR	OUTADR	PRINT IT
4C8A 20 C5		BRA	INST2	RETURN

\*

## \* PRINT PAST EXECUTION INSTRUCTIONS

\*

4C8C BD 42 94	PAST	JSR	PCRLF	PRINT CR & LF
4C8F 7F 40 67		CLR	CNTR	CLEAR COUNT
4C92 BD 4C 63		JSR	TSTTRM	TEST TERM
4C95 27 0B		BEQ	PAST2	
4C97 BD 43 EE		JSR	INDEC	INPUT COUNT
4C9A 25 3E		BCS	PAST6	
4C9C B6 40 5E		LDA A	VALUE+1	GET VALUE
4C9F B7 40 67		STA A	CNTR	SET COUNT
4CA2 FE 40 C0	PAST2	LDX	NXTPC	GET POSITION
4CA5 8C 3D 00	PAST25	CPX	#STRTPC	TABLE BEGINNING?
4CA8 26 03		BNE	PAST3	
4CAA CE 3F 00		LDX	#ENDPC	RESET POINTER
4CAD 09	PAST3	DEX		MOVE BACK ONE INST
4CAE 09		DEX		
4CAF 7A 40 67		DEC	CNTR	DEC THE COUNT
4CB2 26 F1		BNE	PAST25	
4CB4 FF 40 61	PAST4	STX	POINT2	SACE POSITION
4CB7 EE 00		LDX	0, X	GET ADDRESS



4CB9 FF 40 97	STX	OLDPC	SET PC
4CBC BD 42 94	JSR	PCRLF	PRINT CR & LF
4CBF BD 4A 43	JSR	DISO	DISASSEMBLE
4CC2 FE 40 61	LDX	POINT2	GET POINTER
4CC5 08	INX		MOVE TO NEXT
4CC6 08	INX		
4CC7 8C 3F 00	CPX	#ENDPC	END OF TABLE?
4CCA 26 03	BNE	PAST5	
4CCC CE 3D 00	LDX	#STRTPC	RESET POINTER
4CCF BC 40 C0	CPX	NXTPC	FINISHED?
4CD2 26 E0	BNE	PAST4	REPEAT
4CD4 BD 42 94	JSR	PCRLF	PRINT CR & LF
4CD7 7E 41 95	JMP	EXEC	RETURN
4CDA 7E 41 CE	JMP	SYNER	REPORT ERROR

\*

\* GET ADDRESS RANGE

\*

4CDD BD 43 B5	GETRNG	JSR	GETHEX	GET ADDRESS
4CE0 25 2E		BCS	PROT3	ERROR?
4CE2 FF 40 63		STX	FIRST	SAVE VALUE
4CE5 FF 40 65		STX	LAST	
4CE8 BD 43 94		JSR	PRVW	CHECK CHARACTER
4CEB BD 43 D4		JSR	TSTHEX	IS IT HEX?
4CEE 25 08		BCS	GETRN2	
4CF0 BD 43 B5		JSR	GETHEX	GET NEXT ADDRESS
4CF3 25 1B		BCS	PROT3	
4CF5 FF 40 65		STX	LAST	SAVE VALUE
4CF8 39	GETRN2	RTS		RETURN

\*

\* SET PROTECTION FIELDS

\*

4CF9 8D E2	PROT	BSR	GETRNG	GET ADDRESSES
4CFB BD 43 61	PROT2	JSR	NXTCH	GET CHARACTER
4CFE 84 5F		AND A	#\$5F	MAKE UPPER C
4D00 81 58		CMP A	#'X	IS IT EX?
4D02 27 1E		BEQ	PROT4	
4D04 81 57		CMP A	#'W	IS IT WP?
4D06 27 29		BEQ	PROT5	
4D08 81 4D		CMP A	#'M	IS IT MEM?
4D0A 27 34		BEQ	PROT6	
4D0C 81 53		CMP A	#'S	IS IT SIM P?
4D0E 27 03		BEQ	PROT35	
4D10 7E 41 CE	PROT3	JMP	SYNER	REPORT ERROR
4D13 FE 40 C6	PROT35	LDX	SMEND	GET END POINTER
4D16 8C 3F 20		CPX	#EXTAB	FULL?
4D19 27 3C		BEQ	PROT8	
4D1B 8D 40		BSR	SETV	SET VALUES
4D1D FF 40 C6		STX	SMEND	SAVE END
4D20 20 2B		BRA	PROT7	
4D22 FE 40 C8	PROT4	LDX	EXEND	GET END POINTER



4D25	8C	3F	40		CPX	#WPTAB	FULL?
4D28	27	2D			BEQ	PROT8	
4D2A	8D	31			BSR	SETV	SET VALUES
4D2C	FF	40	C8		STX	EXEND	SAVE END
4D2F	20	1C			BRA	PROT7	
4D31	FE	40	CA	PROT5	LDX	WPEND	GET END P
4D34	8C	3F	60		CPX	#MTAB	FULL?
4D37	27	1E			BEQ	PROT8	
4D39	8D	22			BSR	SETV	SET VALUES
4D3B	FF	40	CA		STX	WPEND	SAVE END
4D3E	20	0D			BRA	PROT7	
4D40	FE	40	CC	PROT6	LDX	MEND	GET END P
4D43	8C	3F	80		CPX	#MSGTB	FULL?
4D46	27	0F			BEQ	PROT8	
4D48	8D	13			BSR	SETV	SET VALUES
4D4A	FF	40	CC		STX	MEND	SAVE END
4D4D	BD	43	61	PROT7	JSR	NXTCH	GET NEXT CHAR
4D50	BD	4C	63		JSR	TSTTRM	IS IT TERM?
4D53	26	A4			BNE	PROT	REPEAT?
4D55	20	28			BRA	CMOD35	
4D57	CE	5A	44	PROT8	LDX	#TBLOV	POINT TO STRING
4D5A	7E	41	C9		JMP	ILIN2	GO PRINT IT

\*  
\* SET VALUES  
\*

4D5D	B6	40	63	SETV	LDA A	FIRST	GET FIRST
4D60	F6	40	64		LDA B	FIRST+1	
4D63	8D	06			BSR	STRIT	GO STORE
4D65	B6	40	65		LDA A	LAST	GET LAST
4D68	F6	40	66		LDA B	LAST+1	

\*  
\* STORE VALUES  
\*

4D6B	A7	00		STRIT	STA A	0,X	SAVE CHARACTER
4D6D	E7	01			STA B	1,X	
4D6F	08				INX		BUMP POINTER
4D70	08				INX		
4D71	39				RTS		RETURN

\*  
\* SET MODE  
\*

4D72	BD	4C	2A	CMODE	JSR	EQVAL	GET VALUE
4D75	B6	40	5E		LDA A	VALUE+1	GET VALUE
4D78	81	01			CMP A	#1	CHECK RANGE
4D7A	22	06			BHI	CMODE4	
4D7C	B7	40	9B	CMODE3	STA A	MODE	SET MODE
4D7F	7E	41	95	CMOD35	JMP	EXEC	RETURN
4D82	7E	41	CE	CMODE4	JMP	SYNER	REPORT ERROR



\*  
\* PRINT PROTECTION BOUNDS  
\*

4D85	BD	4C	63	BOUNDS	JSR	TSTTRM	TEST TERM CHAR
4D88	26	16			BNE	BOUND2	
4D8A	BD	42	94		JSR	PCRLF	PRINT CR & LF
4D8D	8D	55			BSR	BOUNDX	PRINT X TAB
4D8F	BD	42	94		JSR	PCRLF	DO CR & LF
4D92	8D	60			BSR	BOUNDW	PRINT W TAB
4D94	BD	42	94		JSR	PCRLF	CR & LF
4D97	8D	6B			BSR	BOUNDM	PRINT M TAB
4D99	BD	42	94		JSR	PCRLF	
4D9C	8D	36			BSR	BOUNS	PRINT S TAB
4D9E	20	2E			BRA	BOUND8	
4DA0	BD	42	94	BOUND2	JSR	PCRLF	PRINT CR & LF
4DA3	BD	43	61		JSR	NXTCH	GET NEXT CHARACTER
4DA6	84	5F			AND A	#\$5F	MASK TO UPPER
4DA8	81	53			CMP A	#'S	IS IT SIM P?
4DAA	27	0F			BEQ	BOUN35	
4DAC	81	58			CMP A	#'X	IS IT EX?
4DAE	27	0F			BEQ	BOUND4	
4DB0	81	57			CMP A	#'W	IS IT WP?
4DB2	27	0F			BEQ	BOUND5	
4DB4	81	4D			CMP A	#'M	IS IT MEM P?
4DB6	27	0F			BEQ	BOUND6	
4DB8	7E	41	CE	BOUND3	JMP	SYNER	REPORT ERROR
4DBB	8D	17		BOUN35	BSR	BOUNS	PRINT S
4DBD	20	0A			BRA	BOUND7	
4DBF	8D	23		BOUND4	BSR	BOUNDX	PRINT X
4DC1	20	06			BRA	BOUND7	
4DC3	8D	2F		BOUND5	BSR	BOUNDW	PRINT W
4DC5	20	02			BRA	BOUND7	
4DC7	8D	3B		BOUND6	BSR	BOUNDM	PRINT M
4DC9	BD	4C	63	BOUND7	JSR	TSTTRM	TEST TERM
4DCC	26	D2			BNE	BOUND2	REPEAT?
4DCE	BD	42	94	BOUND8	JSR	PCRLF	PRINT CR & LF
4DD1	7E	41	95		JMP	EXEC	RETURN
4DD4	CE	5A	68	BOUNS	LDX	#SMPST	POINT TO STRING
4DD7	8D	59			BSR	PBPRT	
4DD9	FE	40	C6		LDX	SMEND	SET END POINT
4DDC	FF	40	CE		STX	TABE	
4DDF	CE	3F	00		LDX	#SMTAB	POINT TO TABLE
4DE2	20	2E			BRA	PRTB	PRINT TABLE
4DE4	CE	5A	59	BOUNDX	LDX	#XPST	POINT TO STRING
4DE7	8D	49			BSR	PBPRT	
4DE9	FE	40	C8		LDX	EXEND	SET END POINTER
4DEC	FF	40	CE		STX	TABE	
4DEF	CE	3F	20		LDX	#EXTAB	POINT TO TABLE
4DF2	20	1E			BRA	PRTB	PRINT TABLE
4DF4	CE	5A	53	BOUNDW	LDX	#WPRST	POINT TO STRING
4DF7	8D	39			BSR	PBPRT	
4DF9	FE	40	CA		LDX	WPEND	SET END P



4DFC FF 40 CE	STX	TABE	
4DFF CE 3F 40	LDX	#WPTAB	POINT TO TABLE
4E02 20 0E	BRA	PRTB	PRINT TABLE
4E04 CE 5A 61	LDX	#MPST	POINT TO STRING
4E07 8D 29	BSR	PBPRT	
4E09 FE 40 CC	LDX	MEND	SET END
4E0C FF 40 CE	STX	TABE	
4E0F CE 3F 60	LDX	#MTAB	POINT TO TABLE

\*  
\* PRINT PROT TABLE  
\*

4E12 BD 42 94	PRTB	JSR	PCRLF	PRINT CR & LF
4E15 BC 40 CE	PRTB1	CPX	TABE	END OF TABLE?
4E18 27 15		BEQ	PRTB2	
4E1A BD 42 94		JSR	PCRLF	PRINT CR & LF
4E1D BD 49 4D		JSR	OUT2SP	
4E20 BD 43 26		JSR	OUTADR	PRINT ADDRESS
4E23 08		INX		BUMP POINTER
4E24 86 2D		LDA A	#'-	OUTPUT '--
4E26 BD 41 09		JSR	PUTCHR	
4E29 BD 43 26		JSR	OUTADR	PRINT ADDRESS
4E2C 08		INX		BUMP TO NEXT
4E2D 20 E6		BRA	PRTB1	REPEAT
4E2F 7E 43 61	PRTB2	JMP	NXTCH	GET NEXT CHAR

\*  
\* PRINT PROTECTION STRING  
\*

4E32 BD 42 85	PBPRT	JSR	PSTRNG	PRINT STRING
4E35 CE 5A 71		LDX	#PRTSTR	POINT TO PROT
4E38 7E 42 87		JMP	PDATA1	PRINT IT

\*  
\* TRAP ON NEST COUNT  
\*

4E3B CE 5A 7D	TRNEST	LDX	#NSTST	POINT TO STRING
4E3E 20 19		BRA	NSTER2	REPORT ERROR

\*  
\* SET TRANSFER TRAP  
\*

4E40 BD 4C 69	STXFR	JSR	TSTON	TEST ON/OFF
4E43 27 05		BEQ	STXFR2	
4E45 7F 40 A6		CLR	XFRT	TURN OFF
4E48 20 36		BRA	RTRN1	RETURN
4E4A 86 01	STXFR2	LDA A	#1	TURN ON
4E4C B7 40 A6		STA A	XFRT	SET FLAG
4E4F 20 2F		BRA	RTRN1	



\*  
\* REPORT XFR TRAP  
\*

4E51 CE 5A B6 XFRTRP LDX #XFRST POINT TO STRING  
4E54 20 03 BRA NSTER2 REPORT

\*  
\* REPORT NEST ERROR  
\*

4E56 CE 5A 89 NSTER LDX #NERST POINT TO STRING  
4E59 7E 45 3E NSTER2 JMP SIMUL7 GO REPORT

\*  
\* PRINT RETURN ADDRESS  
\*

4E5C BD 4C 63 RTRN JSR TSTTRM TEST TERM  
4E5F 26 2A BNE RTRN4 ERROR?  
4E61 7D 40 99 TST NESTC CHECK NEST COUNT  
4E64 27 1D BEQ RTRN2  
4E66 CE 5A 9C LDX #RTADS POINT TO STRING  
4E69 BD 42 85 JSR PSTRNG  
4E6C FE 40 91 LDX SP GET SP  
4E6F 08 INX BACK INTO STACK  
4E70 A6 01 LDA A 1,X GET ADDRESS  
4E72 E6 00 LDA B 0,X  
4E74 B7 40 5E STA A VALUE+1 SAVE ADDRESS  
4E77 F7 40 5D STA B VALUE  
4E7A CE 40 5D LDX #VALUE POINT TO IT  
4E7D BD 43 26 JSR OUTADR PRINT ADDRESS  
4E80 7E 41 95 RTRN1 JMP EXEC RETURN  
4E83 CE 5A A6 RTRN2 LDX #NORTS POINT TO STRING  
4E86 BD 42 85 JSR PSTRNG PRINT IT  
4E89 20 F5 BRA RTRN1  
4E8B 7E 41 CE RTRN4 JMP SYNER REPORT ERROR

\*  
\* DO DISASSEMBLY  
\*

4E8E BD 4C 63 DIS JSR TSTTRM CHECK TERM  
4E91 27 F8 BEQ RTRN4 ERROR?  
4E93 BD 4C DD JSR GETRNG GET ADDRESSES  
4E96 BD 42 94 JSR PCRLF PRINT CR & LF  
4E99 FE 40 63 LDX FIRST GET START ADDR  
4E9C FF 40 97 STX OLDPC SAVE START  
4E9F BD 42 94 DIS2 JSR PCRLF PRINT CR & LF  
4EA2 BD 4A 43 JSR DIS0 DO ONE LINE  
4EA5 B6 40 97 LDA A OLDPC GET NEXT POINTER  
4EA8 F6 40 98 LDA B OLDPC+1  
4EAB B1 40 65 CMP A LAST FINISHED?  
4EAE 22 07 BHI DIS4



4EB0 25 ED		BLO	DIS2	
4EB2 F1 40 66		CMP B	LAST+1	
4EB5 23 E8		BLS	DIS2	
4EB7 FE 40 93	DIS4	LDX	PC	RESET POINTERS
4EBA FF 40 97		STX	OLDPC	
4EBD BD 42 94		JSR	PCRLF	
4EC0 20 BE	DIS6	BRA	RTRN1	RETURN TO EXEC

\*  
\* SET TRACE MODE AND LEVEL  
\*

4EC2 BD 4C 2A	STRAC	JSR	EQVAL	GET VALUE
4EC5 B6 40 5E		LDA A	VALUE+1	
4EC8 B7 40 9D		STA A	TRCF	SAVE IN FLAG
4ECB 20 F3		BRA	DIS6	RETURN

\*  
\* SET STOP ADDRESS  
\*

4ECD BD 4C 5D	SSTOP	JSR	TSTEQ	TEST FOR EQUALS
4ED0 26 0A		BNE	SSTOP2	ERROR?
4ED2 BD 43 B5		JSR	GETHEX	GET ADDRESS
4ED5 25 05		BCS	SSTOP2	ERROR?
4ED7 FF 40 73		STX	END	SAVE ADDRESS
4EDA 20 E4		BRA	DIS6	RETURN
4EDC 20 AD	SSTOP2	BRA	RTRN4	REPORT ERROR

\*  
\* EXAMINE AND CHANGE MEMORY  
\*

4EDE BD 4C 63	XMEM	JSR	TSTTRM	CHECK TERM
4EE1 27 07		BEQ	XMEM2	
4EE3 BD 43 B5		JSR	GETHEX	GET ADDRESS
4EE6 25 F4		BCS	SSTOP2	ERROR?
4EE8 20 03		BRA	XMEM3	
4EEA FE 40 93	XMEM2	LDX	PC	GET DEFAULT ADDR
4EED FF 40 61	XMEM3	STX	POINT2	SAVE POINTER
4EF0 BD 42 94		JSR	PCRLF	PRINT CR & LF
4EF3 CE 40 61	XMEM35	LDX	#POINT2	POINT TO ADDRESS
4EF6 BD 43 26		JSR	OUTADR	PRINT IT
4EF9 BD 49 4F		JSR	OUTSP	PRINT SPACE
4EFC FE 40 61		LDX	POINT2	POINT TO DATA
4EFF BD 43 29		JSR	OUTHEX	PRINT IT
4F02 BD 49 4F		JSR	OUTSP	PRINT SPACE
4F05 FE 40 61		LDX	POINT2	GET ADDRESS
4F08 7F 40 5D		CLR	VALUE	CLEAR OUT NUMBER
4F0B 7F 40 5E		CLR	VALUE+1	
4F0E BD 41 06		JSR	GETCHR	GET CHARACTER
4F11 81 0D		CMP A	#\$D	IS IT CR?
4F13 27 AB		BEQ	DIS6	
4F15 81 0A		CMP A	#\$A	IS IT LF?



4F17 27 24		BEQ	XMEM5	
4F19 81 20	XMEM37	CMP A	##20	IS IT SPACE?
4F1B 27 0D		BEQ	XMEM38	
4F1D BD 43 D4		JSR	TSTHEX	IS IT HEX?
4F20 25 18		BCS	XMEM4	
4F22 BD 43 A0		JSR	WRKHX	WORK IN VALUE
4F25 BD 41 06		JSR	GETCHR	GET NEW CHARACTER
4F28 20 EF		BRA	XMEM37	
4F2A FE 40 61	XMEM38	LDX	POINT2	GET ADDRESS
4F2D B6 40 5E		LDA A	VALUE+1	GET VALUE
4F30 A7 00		STA A	0, X	PUT NEW DATA
4F32 01		NOP		DELAY
4F33 A1 00		CMP A	0, X	CHECK MEMORY
4F35 27 03		BEQ	XMEM4	
4F37 7E 41 C6		JMP	ILIN	REPORT ERROR
4F3A 08	XMEM4	INX		BUMP TO NEXT
4F3B 20 B0		BRA	XMEM3	REPEAT
4F3D 09	XMEM5	DEX		MOVE BACK ONE
4F3E 86 0D		LDA A	##D	OUTPUT CR
4F40 BD 41 09		JSR	PUTCHR	
4F43 FF 40 61		STX	POINT2	SAVE POSITION
4F46 20 AB		BRA	XMEM35	
* SET REGISTERS				
4F48 BD 4C 63	SET	JSR	TSTTRM	CHECK TERM
4F4B 27 3C		BEQ	SET6	ERROR?
4F4D 8D 3D	SET2	BSR	FNDREG	FIND REGISTER
4F4F 25 38		BCS	SET6	ERROR?
4F51 81 01		CMP A	#1	CHECK SIZE
4F53 26 0F		BNE	SET4	
4F55 BD 4C 37		JSR	HEQVAL	GET HEX VALUE
4F58 FE 40 61		LDX	POINT2	GET POINTER
4F5B EE 02		LDX	2, X	POINT TO REG
4F5D B6 40 5E		LDA A	VALUE+1	GET VALUE
4F60 A7 00		STA A	0, X	SET REGISTER
4F62 20 1D		BRA	SET5	
4F64 BD 4C 37	SET4	JSR	HEQVAL	GET HEX VALUE
4F67 FE 40 61		LDX	POINT2	GET POINTER
4F6A EE 02		LDX	2, X	POINT TO REG
4F6C B6 40 5D		LDA A	VALUE	GET VALUE
4F6F F6 40 5E		LDA B	VALUE+1	
4F72 A7 00		STA A	0, X	SET REGISTER
4F74 E7 01		STA B	1, X	
4F76 8C 40 91		CPX	#SP	IS IT SP?
4F79 26 06		BNE	SET5	
4F7B B7 40 81		STA A	MAXSP	SET MAX DEPTH
4F7E F7 40 82		STA B	MAXSP+1	
4F81 BD 4C 63	SET5	JSR	TSTTRM	ANY MORE?
4F84 26 C7		BNE	SET2	
4F86 7E 41 95		JMP	EXEC	RETURN
4F89 7E 41 CE	SET6	JMP	SYNER	REPORT ERROR



\*  
\* FIND REGISTER IN MEMORY  
\*

4F8C BD 43 61	FNDREG	JSR	NXTCH	GET NEXT CHAR
4F8F 25 F8		BCS	SET6	ERROR?
4F91 36		PSH A		SAVE IT
4F92 BD 43 61		JSR	NXTCH	GET NEXT
4F95 32		PUL A		
4F96 24 F1		BCC	SET6	ERROR?
4F98 84 5F	FNDRE1	AND A	##5F	MAKE UPPER
4F9A CE 59 07		LDX	#RGT	POINT TO TABLE
4F9D C6 01		LDA B	#1	SET COUNT
4F9F 6D 00	FNDRE2	TST	0,X	END OF TAB?
4FA1 27 12		BEQ	FNDRE6	
4FA3 A1 00		CMP A	0,X	COMPARE NAMES
4FA5 27 07		BEQ	FNDRE4	MATCH?
4FA7 08		INX		MOVE TO NEXT
4FA8 08		INX		
4FA9 08		INX		
4FAA 08		INX		
4FAB 5C		INC B		BUMP COUNT
4FAC 20 F1		BRA	FNDRE2	REPEAT
4FAE A6 01	FNDRE4	LDA A	1,X	GET SIZE
4FB0 FF 40 61		STX	POINT2	SAVE POSITION
4FB3 0C		CLC		SHOW FOUND
4FB4 39		RTS		RETURN
4FB5 0D	FNDRE6	SEC		SHOW ERROR
4FB6 39		RTS		RETURN

\*  
\* SET BREAK POINT  
\*

4FB7 7F 40 B1	BRK	CLR	BTYP	CLEAR TYPE
4FBA BD 4C 63		JSR	TSTTRM	CHECK TERM
4FBD 27 27		BEQ	BRK3	ERROR?
4FBF 81 40 23		CMP A	#'e' II	CHECK FOR 'e' X
4FC1 26 07		BNE	BRK2	
4FC3 86 03		LDA A	##3	SET DEFAULT TYPE
4FC5 B7 40 B1		STA A	BTYP	
4FC8 20 1F		BRA	BRK4	
4FCA BD 43 61	BRK2	JSR	NXTCH	GET CHARACTER
4FCD 81 40 23		CMP A	#'e' II	IS IT 'e'? X
4FCF 27 18		BEQ	BRK4	
4FD1 BD 51 6C		JSR	FNDTP	FIND TYPE
4FD4 25 10		BCS	BRK3	ERROR?
4FD6 A6 01		LDA A	1,X	GET TYPE
4FD8 BA 40 B1		ORA A	BTYP	MASK IN NEW
4FDB B7 40 B1		STA A	BTYP	SAVE NEW
4FDE 20 EA		BRA	BRK2	REPEAT
4FE0 FE 40 AC	BRK25	LDX	BPPOS	GET BP POS
4FE3 BD 51 4D		JSR	RMVBP	REMOVE BP



4FE6	7E	41	CE	BRK3	JMP	SYNER	REPORT ERROR
4FE9	BD	43	B5	BRK4	JSR	GETHEX	GET ADDRESS
4FEC	25	F8			BCS	BRK3	
4FEE	FF	40	B2		STX	BPAD	SAVE ADDRESS
4FF1	B6	40	B2		LDA A	BPAD	GET ADDRESS
4FF4	F6	40	B3		LDA B	BPAD+1	
4FF7	BD	44	4D		JSR	LKBP	LOOK FOR BREAKPOINT
4FFA	FF	40	AC		STX	BPPOS	SAVE POSITION
4FFD	8C	3D	00		CPX	#STRTPC	OVERFLOW?
5000	26	0F			BNE	BRK5	
5002	CE	5A	C3	BRK42	LDX	#BOVST	POINT TO STRING
5005	BD	42	85		JSR	PSTRNG	PRINT IT
5008	FE	40	AC		LDX	BPPOS	GET POS
500B	BD	51	4D		JSR	RMVBP	REMOVE IT
500E	7E	41	95	BRK45	JMP	EXEC	RETURN
5011	C6	08		BRK5	LDA B	#8	SET COUNTER
5013	6F	00		BRK55	CLR	0,X	CLEAR OUT BP
5015	08				INX		
5016	5A				DEC B		DEC THE COUNT
5017	26	FA			BNE	BRK55	
5019	FE	40	AC		LDX	BPPOS	GET POSITION
501C	B6	40	B2		LDA A	BPAD	GET ADDRESS
501F	F6	40	B3		LDA B	BPAD+1	
5022	A7	00			STA A	0,X	PUT IN ADDRESS
5024	E7	01			STA B	1,X	
5026	B6	40	B1		LDA A	BTYP	GET TYPE
5029	A7	07			STA A	7,X	PUT IN
502B	BD	4C	63		JSR	TSTTRM	CHECK TERM
502E	27	31			BEQ	BRK7	
5030	BD	43	61		JSR	NXTCH	GET CHARACTER
5033	81	3E			CMP A	#' >	CHECK COUNTERS
5035	27	14			BEQ	BRK6	
5037	81	3C			CMP A	#' <	
5039	27	09			BEQ	BRK58	
503B	FE	40	50		LDX	BUFPNT	RESET POINTER
503E	09				DEX		
503F	FF	40	50		STX	BUFPNT	
5042	20	1D			BRA	BRK7	
5044	FE	40	AC	BRK58	LDX	BPPOS	GET POSITION
5047	86	80			LDA A	#80	SET BEFORE COUNT
5049	A7	02			STA A	2,X	
504B	BD	43	EE	BRK6	JSR	INDEC	GET COUNT
504E	25	90			BCS	BRK25	ERROR?
5050	FE	40	AC		LDX	BPPOS	GET POSITION
5053	B6	40	5D		LDA A	VALUE	GET VALUE
5056	2B	88			BMI	BRK25	COUNT TOO BIG?
5058	F6	40	5E		LDA B	VALUE+1	
505B	E7	03			STA B	3,X	SAVE COUNT
505D	AA	02			ORA A	2,X	
505F	A7	02			STA A	2,X	
5061	FE	40	AE	BRK7	LDX	BPEND	POINT TO BP
5064	BC	40	AC		CPX	BPPOS	CHECK IF NEW
5067	26	08			BNE	BRK72	
5069	C6	08			LDA B	#8	



506B	BD	44	2D	JSR	ADDBX	CALCULATE END
506E	FF	40	AE	STX	BPEND	SAVE END
5071	B6	40	B1	LDA A	BTYP	GET TYPE
5074	85	20		BIT A	#\$20	IS IT HIST?
5076	26	0B		BNE	DOHB	
5078	85	40		BIT A	#\$40	IS IT MESSAGE?
507A	26	12		BNE	DOMB	
507C	85	80		BIT A	#\$80	IS IT JUMP?
507E	26	45		BNE	DOJB	
5080	7E	50	F1	JMP	BEXP	PROCESS EXP

\*  
\* SETUP HISTOGRAM COUNT  
\*

5083	BD	4C	63	DOHB	JSR	TSTTRM	CHECK TERM
5086	27	03		BEQ	DOHB2		
5088	7E	4F	E0	DOHB1	JMP	BRK25	REPORT ERROR
508B	7E	41	95	DOHB2	JMP	EXEC	RETURN

\*  
\* SETUP MESSAGE  
\*

508E	FE	40	B6	DOMB	LDX	NXTMSG	CHECK FULL
5091	8C	3F	A0		CPX	#USER	
5094	26	03			BNE	DOMB1	
5096	7E	50	02		JMP	BRK42	
5099	B6	40	B6	DOMB1	LDA A	NXTMSG	GET POINTER
509C	F6	40	B7		LDA B	NXTMSG+1	
509F	FE	40	AC		LDX	BPPOS	POINT TO BP
50A2	A7	05			STA A	5,X	SAVE POINTER
50A4	E7	06			STA B	6,X	
50A6	FE	40	B6		LDX	NXTMSG	POINT TO SPACE
50A9	BD	43	61	DOMB2	JSR	NXTCH	GET NEXT CHAR
50AC	81	0D			CMP A	#\$D	IS IT CR?
50AE	27	0B			BEQ	DOMB4	
50B0	A7	00			STA A	0,X	SAVE CHARACTER
50B2	08				INX		
50B3	8C	3F	A0		CPX	#USER	TABLE FULL?
50B6	26	F1			BNE	DOMB2	
50B8	7E	50	02		JMP	BRK42	
50BB	86	04		DOMB4	LDA A	#4	SETUP TERM
50BD	A7	00			STA A	0,X	
50BF	08				INX		FIX POINTER
50C0	FF	40	B6		STX	NXTMSG	MARK END
50C3	20	C6			BRA	DOHB2	RETURN

\*  
\* SETUP JUMP BP  
\*

50C5	BD	43	B5	DOJB	JSR	GETHEX	GET ADDRESS
50C8	25	BE		BCS	DOHB1		ERROR?



50CA B6 40 5D	LDA A	VALUE	GET VALUE
50CD F6 40 5E	LDA B	VALUE+1	
50D0 FE 40 AC	LDX	BPP05	POINT TO BP
50D3 A7 05	STA A	5,X	SAVE ADDRESS
50D5 E7 06	STA B	6,X	
50D7 20 B2	BRA	DOHB2	RETURN

\*  
\* SET CONDITIONAL STATUS  
\*

50D9 FE 40 AC	SETCN	LDX	BPP05	POINT TO BP
50DC C6 01		LDA B	#1	
50DE B6 40 68		LDA A	LSTTRM	GET TERM
50E1 81 21		CMP A	#1	NOT EQUALS?
50E3 26 04		BNE	SETCN2	
50E5 5C		INC B		SET IND
50E6 BD 43 61		JSR	NXTCH	GET NEXT CHAR
50E9 81 3D	SETCN2	CMP A	#'=	IS IT '='?
50EB 26 01		BNE	SETCN4	ERROR?
50ED 39		RTS		RETURN
50EE 7E 4F E0	SETCN4	JMP	BRK25	REPORT ERROR

\*  
\* PROCESS BP EXPRESSION  
\*

50F1 BD 4C 63	BEXP	JSR	TSTTRM	CHECK TERM
50F4 27 95		BEQ	DOHB2	
50F6 BD 43 61		JSR	NXTCH	GET CHARACTER
50F9 84 5F		AND A	#\$5F	MAKE UPPER
50FB 81 49		CMP A	#'I	IS IT 'I'?
50FD 26 EF		BNE	SETCN4	ERROR?
50FF BD 43 61		JSR	NXTCH	GET NEXT
5102 84 5F		AND A	#\$5F	MAKE UPPER
5104 81 46		CMP A	#'F	IS IT 'F'?
5106 26 E6		BNE	SETCN4	
5108 BD 43 7E		JSR	SKPSPC	SKIP SPACES
510B BD 43 61		JSR	NXTCH	GET CHARACTER
510E 24 12		BCC	BEXP2	LETTER?
5110 81 24		CMP A	#'\$	IS IT '\$'?
5112 26 DA		BNE	SETCN4	ERROR?
5114 BD 43 B5		JSR	GETHEX	GET ADDRESS
5117 25 D5		BCS	SETCN4	
5119 BD 50 D9		JSR	SETCN	SET CONDITION
511C CA 80		ORA B	#\$80	SET ADDR TYPE
511E E7 04		STA B	4,X	SAVE IN BP
5120 20 1C		BRA	BEXP4	
5122 BD 4F 98	BEXP2	JSR	FNDRE1	FIND REGISTER
5125 25 C7	BEXP3	BCS	SETCN4	ERROR?
5127 58		ASL B		FIX SPEC
5128 58		ASL B		
5129 58		ASL B		
512A 58		ASL B		



512B F7 40 9A	STA B	OPCNT	SAVE VALUE
512E BD 43 61	JSR	NXTCH	GET CHARACTER
5131 BD 50 D9	JSR	SETCN	GET CONDITIONAL
5134 FA 40 9A	ORA B	OPCNT	
5137 E7 04	STA B	4,X	SAVE IN BP
5139 BD 43 B5	JSR	GETHEX	GET VALUE
513C 25 E7	BCS	BEXP3	ERROR?
513E B6 40 5D	BEXP4	LDA A	VALUE
5141 F6 40 5E	LDA B	VALUE+1	
5144 FE 40 AC	LDX	BPPOS	POINT TO BP
5147 A7 05	STA A	5,X	SAVE VALUE
5149 E7 06	STA B	6,X	
514B 20 4A	BRA	HIST7	RETURN

\*  
 \* REMOVE BREAKPOINT FROM TABLE  
 \*

514D BC 40 AE	RMVBP	CPX	BPEND	END OF LIST?
5150 27 19		BEQ	RMVBP4	
5152 BC 40 AE	RMVBP1	CPX	BPEND	END OF TABLE?
5155 27 07		BEQ	RMVBP2	
5157 A6 08		LDA A	8,X	GET CHARACTER
5159 A7 00		STA A	0,X	MOVE IT DOWN
515B 08		INX		BUMP TO NEXT
515C 20 F4		BRA	RMVBP1	REPEAT
515E B6 40 AF	RMVBP2	LDA A	BPEND+1	GET END POINT
5161 80 08		SUB A	#8	DEC BY 8
5163 B7 40 AF		STA A	BPEND+1	SAVE NEW
5166 24 03		BCC	RMVBP4	
5168 7A 40 AE		DEC	BPEND	ADJUST MSB
516B 39	RMVBP4	RTS		RETURN

\*  
 \* FIND BP TYPE IN TABLE  
 \*

516C CE 59 24	FNDTP	LDX	#TYPTB	POINT TO TABLE
516F 6D 00	FNDTP2	TST	0,X	END OF TABLE?
5171 27 0A		BEQ	FNDTP6	
5173 A1 00		CMP A	0,X	CHECK CHARACTER
5175 27 04		BEQ	FNDTP4	MATCH?
5177 08		INX		MOVE TO NEXT
5178 08		INX		
5179 20 F4		BRA	FNDTP2	
517B 0C	FNDTP4	CLC		SHOW MATCH
517C 39		RTS		RETURN
517D 0D	FNDTP6	SEC		SHOW ERROR
517E 39		RTS		RETURN

\*  
 \* PRINT HISTOGRAMS  
 \*



517F	8D	19	HIST	BSR	SETFL	SET UP POINTERS
5181	BD	51	CE	HIST2	JSR	GET BP POS
5184	26	0E		BNE	HIST6	FINISHED?
5186	A6	07		LDA A	7,X	GET TYPE
5188	84	20		AND A	#\$20	IS IT HIST?
518A	27	F5		BEQ	HIST2	
518C	8D	28		BSR	PBPAD	PRINT ADDRESS
518E	5F			CLR B		CLEAR FLAG
518F	BD	42	D5	JSR	OUTDEC	OUTPUT COUNT
5192	20	ED		BRA	HIST2	REPEAT
5194	BD	42	94	HIST6	JSR	PCRLF
5197	7E	41	95	HIST7	JMP	EXEC
						RETURN

\*  
\* SETUP FIRST & LAST POINTERS  
\*

519A	BD	42	94	SETFL	JSR	PCRLF	PRINT CR & LF
519D	CE	00	00	LDX	#0		SET FIRST
51A0	FF	40	63	STX	FIRST		
51A3	09			DEX			SET LAST
51A4	FF	40	65	STX	LAST		
51A7	BD	4C	63	JSR	TSTTRM		CHECK TERM
51AA	27	03		BEQ	SETFL2		
51AC	BD	4C	DD	JSR	GETRNG		GET RANGE
51AF	CE	3B	F8	SETFL2	LDX	#BPTAB-8	
51B2	FF	40	B4	STX	NXTBP		SET POINTER
51B5	39			RTS			RETURN

\*  
\* PRINT BP ADDRESS  
\*

51B6	BD	42	94	PBPAD	JSR	PCRLF	PRINT CR & LF
51B9	BD	49	4F		JSR	OUTSP	PRINT SPACE
51BC	BD	43	26		JSR	OUTADR	PRINT ADDRESS
51BF	CE	5A	E4		LDX	#DSHST	POINT TO STRING
51C2	BD	42	87		JSR	PDATA1	PRINT IT
51C5	FE	40	B4		LDX	NXTBP	POINT TO BP
51C8	08				INX		BUMP TO INFO
51C9	08				INX		
51CA	08				INX		
51CB	08				INX		
51CC	08				INX		
51CD	39				RTS		RETURN

\*  
\* RETURN NEXT BP IN RANGE  
\*

51CE	FE	40	B4	NXTRB	LDX	NXTBP	POINT TO BP
51D1	C6	08			LDA B	#8	ADD IN 8
51D3	BD	44	2D		JSR	ADDBX	
51D6	FF	40	B4	NXTRB2	STX	NXTBP	SAVE NEW POS



51D9 BC 40 AE	CPX	BPEND	END OF LIST?
51DC 27 0D	BEQ	NXTRB4	
51DE A6 00	LDA A	0,X	GET ADDRESS
51E0 E6 01	LDA B	1,X	
51E2 8D 0A	BSR	INRNG	CHECK RANGE
51E4 26 E8	BNE	NXTRB	REPEAT
51E6 FE 40 B4	LDX	NXTBP	POINT TO BP
51E9 4F	CLR A		SHOW MATCH
51EA 39	RTS		RETURN
51EB 86 01	NXTRB4 LDA A	#1	SHOW ERROR
51ED 39	RTS		RETURN

\*  
\* ADDRESS IN RANGE CHECK  
\*

51EE CE 40 63	INRNG	LDX	#FIRST	POINT TO 1ST
51F1 8D 0B	BSR	DCMP		COMPARE IT
51F3 25 F6	BLO	NXTRB4		
51F5 CE 40 65	LDX	#LAST		POINT TO LAST
51F8 8D 04	BSR	DCMP		COMPARE
51FA 22 EF	BHI	NXTRB4		
51FC 4F	CLR A			SHOW MATCH
51FD 39	RTS			RETURN

\*  
\* DO 16 BIT COMPARE  
\*

51FE A1 00	DCMP	CMP A	0,X	CHECK MSB
5200 26 02		BNE	DCMP2	
5202 E1 01		CMP B	1,X	COMPARE LSB
5204 39	DCMP2	RTS		RETURN

\*  
\* PRINT BREAK POINTS  
\*

5205 BD 51 9A	PBP	JSR	SETFL	SET RANGES
5208 8D C4	PBP2	BSR	NXTRB	GET BREAKPOINT
520A 26 1E		BNE	PBP6	FINISHED?
520C BD 51 B6		JSR	PBPAD	PRINT ADDRESS
520F E6 02		LDA B	2,X	GET TYPE
5211 CE 59 24		LDX	#TYPTB	POINT TO TABLE
5214 86 08		LDA A	#8	SET COUNTER
5216 B7 40 67		STA A	CNTR	
5219 56	PBP3	ROR B		CHECK BIT BY BIT
521A 24 05		BCC	PBP4	
521C A6 00		LDA A	0,X	GET CHARACTER
521E BD 41 09		JSR	PUTCHR	PRINT IT
5221 08	PBP4	INX		MOVE TO NEXT ENTRY
5222 08		INX		
5223 7A 40 67		DEC	CNTR	DEC THE COUNT
5226 26 F1		BNE	PBP3	REPEAT



```

5228 20 DE          BRA    PBP2
522A 7E 51 94      PBP6    JMP    HIST6    RETURN
*
* CLEAR BREAKPOINTS
*
522D BD 51 9A      CLB     JSR     SETFL    SET RANGE
5230 8D 9C          CLB2    BSR     NXTRB    GET BP ADDRESS
5232 26 12          BNE     CLB4    FINISHED?
5234 BD 51 4D          JSR     RMVBP    REMOVE BP
5237 B6 40 B5          LDA A    NXTBP+1  GET POINTER
523A 80 08          SUB A     #8      DEC BY 8
523C B7 40 B5          STA A    NXTBP+1  SAVE RESULT
523F 24 EF          BCC     CLB2
5241 7A 40 B4          DEC     NXTBP    ADJUST MSB
5244 20 EA          BRA     CLB2    REPEAT
5246 7E 41 95      CLB4    JMP     EXEC    RETURN

```

```

*
* CLEAR HISTOGRAM COUNTERS
*

```

```

5249 BD 51 9A      CLH     JSR     SETFL    SET RANGE
524C BD 51 CE      CLH2    JSR     NXTRB    GET BREAKPOINT
524F 26 F5          BNE     CLB4    FINISHED?
5251 6F 05          CLR     5,X    CLEAR COUNTER
5253 6F 06          CLR     6,X
5255 20 F5          BRA     CLH2    REPEAT

```

```

*
* DO COUNTED SIMULATION
*

```

```

5257 86 FF          TCSIM   LDA A    ##FF    SET TRACE ON
5259 B7 40 9C          STA A    PRON
525C 20 03          BRA     CSIM2

```

```

525E 7F 40 9C      CSIM    CLR     PRON    TURN TRACE OFF
5261 CE 00 01      CSIM2   LDX     #1      SET DEFAULT COUNT
5264 FF 40 87          STX     SIMCNT
5267 BD 4C 63          JSR     TSTTRM    TEST TERM
526A 27 08          BEQ     CSIM4
526C BD 43 EE          JSR     INDEC    GET COUNT
526F 25 06          BCS     CSIM6    ERROR?
5271 FF 40 87          STX     SIMCNT    SET COUNT
5274 7E 44 90      CSIM4   JMP     SIMUL2  GO SIMULATE
5277 7E 41 CE      CSIM6   JMP     SYNER    REPORT ERROR

```

```

*
* PRINT STATES COUNTER
*

```

```

527A CE 59 9F      PSTAT   LDX     #STST    POINT TO STRING

```



527D BD 42 85	JSR	PSTRNG	PRINT IT
5280 CE 40 A7	LDX	#STATES	POINT TO COUNTER
5283 BD 43 26	JSR	OUTADR	PRINT IT
5286 08	INX		
5287 BD 43 26	JSR	OUTADR	
528A 20 BA	PSTAT4	BRA	CLB4
			RETURN

\*  
\* SET DELAY  
\*

528C BD 4C 2A	SDELY	JSR	EQVAL	CHECK VALUE
528F B6 40 5E		LDA A	VALUE+1	GET VALUE
5292 B7 40 8B		STA A	DELAY	SET DELAY
5295 20 F3		BRA	PSTAT4	

\*  
\* SET INDIRECTION MODE  
\*

5297 BD 4C 69	SIND	JSR	TSTON	TEST ON/OFF
529A 26 07		BNE	SIND2	
529C 86 FF		LDA A	##FF	SET TO ON
529E B7 40 A1		STA A	VFLG	
52A1 20 E7		BRA	PSTAT4	RETURN
52A3 7F 40 A1	SIND2	CLR	VFLG	CLEAR MODE
52A6 20 E2		BRA	PSTAT4	

\*  
\* SET INTERRUPT STATUS  
\*

52A8 BD 4C 2A	SIRQ	JSR	EQVAL	GET VALUE
52AB FF 40 7D		STX	IRQC	SET COUNTERS
52AE FF 40 7F		STX	IRQC2	
52B1 20 D7		BRA	PSTAT4	

52B3 BD 4C 2A	SNMI	JSR	EQVAL	GET VALUE
52B6 FF 40 79		STX	NMIC	SET COUNTERS
52B9 FF 40 7B		STX	NMIC2	
52BC 20 CC		BRA	PSTAT4	

\*  
\* HEX CALCULATOR  
\*

52BE 7F 40 B1	CALC	CLR	BTYP	CLEAR MODE
52C1 CE 59 38		LDX	#CLCP	POINT TO PROMPT
52C4 BD 42 85		JSR	PSTRNG	PRINT IT
52C7 BD 42 3C		JSR	INBUF	GET INPUT LINE
52CA BD 43 94		JSR	PRVW	CHECK FOR CR
52CD 81 0D		CMP A	##D	
52CF 27 51		BEG	CALC9	
52D1 8D 51		BSR	CNUM	GET FIRST NUMBER



52D3 FF 40 63	STX	FIRST	SAVE IT
52D6 BD 4C 63	JSR	TSTTRM	END OF LINE?
52D9 27 2F	BEQ	CALC8	
52DB 81 2B	CMP A	#'+	IS IT ADD?
52DD 27 07	BEQ	CALC5	
52DF 81 2D	CMP A	#'-	IS IT SUBTRACT?
52E1 26 56	BNE	CNUM6	
52E3 B7 40 B1	STA A	BTYP	SET MODE
52E6 8D 3C	BSR	CNUM	GET SECOND
52E8 FF 40 65	STX	LAST	SAVE IT
52EB B6 40 63	LDA A	FIRST	GET NUMBER
52EE F6 40 64	LDA B	FIRST+1	
52F1 7D 40 B1	TST	BTYP	ADD?
52F4 26 08	BNE	CALC6	
52F6 FB 40 66	ADD B	LAST+1	ADD NUMBERS
52F9 B9 40 65	ADC A	LAST	
52FC 20 06	BRA	CALC7	
52FE F0 40 66	SUB B	LAST+1	SUBTRACT NUMBERS
5301 B2 40 65	SBC A	LAST	
5304 B7 40 63	STA A	FIRST	SAVE RESULT
5307 F7 40 64	STA B	FIRST+1	
530A BD 42 94	JSR	PCRLF	PRINT CR & LF
530D 86 24	LDA A	#'\$	OUTPUT '\$'
530F BD 41 09	JSR	PUTCHR	
5312 CE 40 63	LDX	#FIRST	POINT TO NUMBER
5315 BD 43 26	JSR	OUTADR	PRINT HEX
5318 BD 49 4D	JSR	OUT2SP	PRINT SPACES
531B 09	DEX		ADJUST POINTER
531C 5F	CLR B		SET FLAG
531D BD 42 D5	JSR	OUTDEC	PRINT DECIMAL
5320 20 9C	BRA	CALC	REPEAT
5322 20 60	BRA	DOASM4	RETURN

\*  
\* GET NUMBER FOR CALC  
\*

5324 BD 43 94	CNUM	JSR	PRVW	CHECK CHARACTER
5327 81 24		CMP A	#'\$	IS IT '\$'?
5329 26 0A		BNE	CNUM2	
532B BD 43 61		JSR	NXTCH	EAT CHARACTER
532E BD 43 B5		JSR	GETHEX	INPUT HEX
5331 25 06		BCS	CNUM6	ERROR?
5333 20 03		BRA	CNUM4	
5335 BD 43 EE	CNUM2	JSR	INDEC	GET DECIMAL
5338 39	CNUM4	RTS		RETURN
5339 20 4C	CNUM6	BRA	DOASM6	REPORT ERROR

\*  
\* PRINT STACK CONTENTS  
\*

533B BD 43 EE	PSTK	JSR	INDEC	GET COUNT
533E 25 F9		BCS	CNUM6	ERROR?



5340 B6 40 5E	LDA A	VALUE+1	GET NUMBER
5343 26 02	BNE	PSTK0	
5345 86 06	LDA A	#6	SET DEFAULT
5347 FE 40 91	LDX	SP	GET S POINTER
534A 08	PSTK1	INX	BUMP POINTER
534B 4A	DEC A		DEC THE COUNT
534C 26 FC	BNE	PSTK1	
534E BC 40 91	CPX	SP	FINISHED?
5351 27 CF	BEQ	CALC9	
5353 BD 42 94	JSR	PCRLF	PRINT CR & LF
5356 BD 49 4F	JSR	OUTSP	PRINT SPACE
5359 BD 43 29	JSR	OUTHEX	PRINT CONTENTS
535C 09	DEX		DEC POINTER
535D 20 EF	BRA	PSTK2	REPEAT

\*

\* DO LINE ASSEMBLY

\*

535F BD 43 B5	DOASM	JSR	GETHEX	GET ADDRESS
5362 25 23		BCS	DOASM6	ERROR?
5364 FF 40 5F		STX	POINT	SAVE POINTER
5367 BD 42 94	DOASM2	JSR	PCRLF	PRINT CR & LF
536A BD 49 4F		JSR	OUTSP	PRINT SPACE
536D CE 40 5F		LDX	#POINT	POINT TO ADDRESS
5370 BD 43 26		JSR	OUTADR	PRINT IT
5373 BD 49 4F		JSR	OUTSP	PRINT SPACE
5376 BD 42 3C		JSR	INBUF	GET LINE
5379 BD 43 7E		JSR	SKPSPC	SKIP SPACES
537C 81 0D		CMP A	##D	IS IT NULL?
537E 27 04		BEQ	DOASM4	
5380 8D 08		BSR	ASML	ASMBL LINE
5382 20 E3		BRA	DOASM2	REPEAT
5384 7E 41 95	DOASM4	JMP	EXEC	RETURN
5387 7E 41 CE	DOASM6	JMP	SYNER	REPORT ERROR

\*

\* DO ASSEMBLY

\*

538A 86 50	ASML	LDA A	##50	SET DEFAULT INHER
538C B7 40 B1		STA A	BTYP	
538F 7F 40 6F		CLR	MOD	CLEAR MODE
5392 CE 40 70		LDX	#OPND	POINT TO STORAGE
5395 BD 43 61		JSR	NXTCH	GET MNEMONIC
5398 84 5F		AND A	##5F	MAKE UPPER
539A A7 00		STA A	0, X	SAVE IN MEMORY
539C BD 43 61		JSR	NXTCH	
539F 84 5F		AND A	##5F	
53A1 A7 01		STA A	1, X	
53A3 BD 43 61		JSR	NXTCH	
53A6 84 5F		AND A	##5F	MAKE UPPER
53A8 A7 02		STA A	2, X	
53AA BD 43 94		JSR	PRVW	CHECK CHARACTER



53AD 81 0D		CMP A	##D	IS IT CR?
53AF 27 56		BEQ	ASML7	
53B1 BD 43 42		JSR	CLASS	CLASSIFY CHAR
53B4 25 18		BCS	ASML4	
53B6 84 5F		AND A	##5F	MAKE UPPER CASE
53B8 81 41		CMP A	#'A	IS IT 'A'?
53BA 26 05		BNE	ASML3	
53BC 7C 40 6F		INC	MOD	SET MOD
53BF 20 0A		BRA	ASML35	
53C1 81 42	ASML3	CMP A	#'B	IS IT 'B'?
53C3 26 09		BNE	ASML4	
53C5 7C 40 6F		INC	MOD	
53C8 7C 40 6F		INC	MOD	SET MOD
53CB BD 43 61	ASML35	JSR	NXTCH	GET CHARACTER
53CE BD 43 7E	ASML4	JSR	SKPSPC	SKIP SPACES
53D1 81 0D		CMP A	##D	END OF LINE?
53D3 27 32		BEQ	ASML7	
53D5 BD 43 94		JSR	PRVW	CHECK CHARACTER
53D8 81 23		CMP A	#'#	IS IT '#'?
53DA 27 19		BEQ	ASML6	
53DC BD 53 24		JSR	CNUM	GET NUMBER
53DF B6 40 68		LDA A	LSTTRM	CHECK TERM
53E2 81 2C		CMP A	#','	IS IT COMMA
53E4 26 0B		BNE	ASML5	IF SO, INDEXED
53E6 BD 43 61		JSR	NXTCH	GET CHARACTER
53E9 81 58		CMP A	#'X	IS IT X?
53EB 26 9A		BNE	DOASM6	ERROR?
53ED 86 30		LDA A	##30	SET TYPE
53EF 20 13		BRA	ASML65	
53F1 86 40	ASML5	LDA A	##40	SET EXTENDED
53F3 20 0F		BRA	ASML65	
53F5 BD 43 61	ASML6	JSR	NXTCH	EAT '#'
53F8 BD 43 94		JSR	PRVW	CHECK NEXT
53FB 81 27		CMP A	#'"	IS IT '"?"
53FD 27 29		BEQ	ASML8	
53FF BD 53 24		JSR	CNUM	GET NUMBER
5402 86 10	ASML63	LDA A	##10	SET IMMEDIATE
5404 B7 40 B1	ASML65	STA A	BTYP	SAVE TYPE
5407 BD 54 36	ASML7	JSR	FNDMN	FIND CODE
540A FE 40 5F		LDX	POINT	GET POINTER
540D A7 00		STA A	0,X	STUFF OPCODE
540F C1 10		CMP B	##10	
5411 27 10		BEQ	ASML77	
5413 C1 30		CMP B	##30	
5415 26 06		BNE	ASML75	
5417 08		INX		BUMP POINTER
5418 B6 40 5D		LDA A	VALUE	GET NUMBER
541B A7 00		STA A	0,X	
541D 08	ASML75	INX		BUMP POINTER
541E B6 40 5E		LDA A	VALUE+1	GET LSB
5421 A7 00		STA A	0,X	
5423 08	ASML77	INX		BUMP ONE MORE
5424 FF 40 5F		STX	POINT	SAVE POS
5427 39		RTS		RETURN



5428	BD	43	61	ASML8	JSR	NXTCH	EAT "/"
542B	BD	43	61		JSR	NXTCH	GET ASCII CHAR
542E	7F	40	5D		CLR	VALUE	
5431	B7	40	5E		STA A	VALUE+1	SAVE VALUE
5434	20	CC			BRA	ASML63	

\*  
\* FIND MNEMONIC  
\*

5436	CE	5B	00	FNDMN	LDX	#OPTAB	SET POINTER
5439	7F	40	67		CLR	CNTR	
543C	FF	40	C2	FNDMN1	STX	OPPNT	SAVE POS
543F	B6	40	70		LDA A	OPND	GET CHARACTERS
5442	F6	40	71		LDA B	OPND+1	
5445	A1	00			CMP A	0,X	COMPARE TO TABLE
5447	26	0D			BNE	FNDMN2	
5449	E1	01			CMP B	1,X	
544B	26	09			BNE	FNDMN2	
544D	36				PSH A		SAVE CHAR
544E	B6	40	72		LDA A	OPND+2	GET 3RD CHAR
5451	A1	02			CMP A	2,X	COMPARE
5453	32				PUL A		RESTORE
5454	27	0C			BEQ	FNDMN4	
5456	08			FNDMN2	INX		MOVE TO NEXT ENTRY
5457	08				INX		
5458	08				INX		
5459	08				INX		
545A	7C	40	67		INC	CNTR	
545D	26	DD			BNE	FNDMN1	FINISHED?
545F	7E	41	CE		JMP	SYNER	REPORT ERROR
5462	F6	40	67	FNDMN4	LDA B	CNTR	GET OPCODE
5465	CE	5F	00		LDX	#OPTAB+1024	
5468	BD	44	2D		JSR	ADDBX	ADD TO POINTER
546B	FF	40	C4		STX	OPPNT2	SAVE POS
546E	E6	00			LDA B	0,X	GET CODE
5470	C4	0F			AND B	##F	MASK BITS
5472	27	0E			BEQ	FNDM45	
5474	C1	02			CMP B	#2	MODIFIER?
5476	22	0A			BHI	FNDM45	
5478	F1	40	6F		CMP B	MOD	CHECK MODIFIER
547B	27	0A			BEQ	FNDMN5	
547D	FE	40	C2	FNDM42	LDX	OPPNT	SET POINTER
5480	20	D4			BRA	FNDMN2	REPEAT
5482	7D	40	6F	FNDM45	TST	MOD	CHECK MODIFIER
5485	26	F6			BNE	FNDM42	
5487	C1	07		FNDMN5	CMP B	#7	CHECK TYPE
5489	24	11			BHS	FNDMN6	
548B	C1	03			CMP B	#3	
548D	27	18			BEQ	FNDMN7	RELATIVE TYPE?
548F	C1	04			CMP B	#4	
5491	27	14			BEQ	FNDMN7	
5493	E6	00			LDA B	0,X	GET CODE
5495	C4	70			AND B	##70	GET ADDRESS TYPE



5497	F1	40	B1		CMP B	BTYP	COMPARE
549A	26	E1			BNE	FNDM42	
549C	FE	40	C2	FNDMN6	LDX	OPPNT	GET POINTER
549F	E6	03			LDA B	3,X	GET COUNTS
54A1	C4	30			AND B	##30	GET BYTE COUNT
54A3	B6	40	67		LDA A	CNTR	GET OPCODE
54A6	39				RTS		RETURN
54A7	FE	40	5F	FNDMN7	LDX	POINT	GET ADDRESS
54AA	08				INX		BUMP BY 2
54AB	08				INX		
54AC	FF	40	61		STX	POINT2	SAVE NEW ADDR
54AF	B6	40	5D		LDA A	VALUE	GET REL ADDRESS
54B2	F6	40	5E		LDA B	VALUE+1	
54B5	F0	40	62		SUB B	POINT2+1	SUBTRACT VALUES
54B8	B2	40	61		SBC A	POINT2	
54BB	B7	40	5D		STA A	VALUE	SAVE RESULT
54BE	F7	40	5E		STA B	VALUE+1	
54C1	20	D9			BRA	FNDMN6	FINISH UP

\*  
\* SET I TRAP  
\*

54C3	BD	40	69	ITRAP	JSR	TSTON	CHECK ON/OFF
54C6	26	07			BNE	ITRAP2	
54C8	86	FF			LDA A	##FF	SET ON
54CA	B7	40	AB		STA A	ITRF	
54CD	20	03			BRA	ITRAP4	
54CF	7F	40	AB	ITRAP2	CLR	ITRF	TURN OFF
54D2	7E	41	95	ITRAP4	JMP	EXEC	RETURN

\*  
\* CLEAR PROTECTION TABLES  
\*

54D5	BD	40	63	CLP	JSR	TSTTRM	CHECK TERM
54D8	26	06			BNE	CLP2	
54DA	BD	41	7C		JSR	CLRPR	CLEAR ALL TABLES
54DD	7E	41	95	CLP1	JMP	EXEC	RETURN
54E0	BD	43	61	CLP2	JSR	NXTCH	GET CHARACTER
54E3	84	5F			AND A	##5F	MAKE UPPER CASE
54E5	81	53			CMP A	##'S	IS IT S?
54E7	27	0F			BEQ	CLP4	
54E9	81	58			CMP A	##'X	IS IT X?
54EB	27	13			BEQ	CLP5	
54ED	81	57			CMP A	##'W	IS IT W?
54EF	27	17			BEQ	CLP6	
54F1	81	4D			CMP A	##'M	IS IT M?
54F3	27	1B			BEQ	CLP7	
54F5	7E	41	CE		JMP	SYNER	REPORT ERROR
54F8	CE	3F	00	CLP4	LDX	##SMTAB	POINT TO SIM TAB
54FB	FF	40	C6		STX	SMEND	SET END
54FE	20	16			BRA	CLP8	
5500	CE	3F	20	CLP5	LDX	##EXTAB	POINT TO EX TAB



5503	FF 40 C8	STX	EXEND	SET END
5506	20 0E	BRA	CLP8	
5508	CE 3F 40	CLP6	LDX	#WPTAB POINT TO WP TAB
550B	FF 40 CA	STX	WPEND	SET END
550E	20 06	BRA	CLP8	
5510	CE 3F 60	CLP7	LDX	#MTAB POINT TO MEM TAB
5513	FF 40 CC	STX	MEND	
5516	BD 43 61	CLP8	JSR	NXTCH GET CHARACTER
5519	81 0D	CMP A	#\$D	IS IT TERM?
551B	27 C0	BEQ	CLP1	
551D	20 C1	BRA	CLP2	REPEAT

\*

\* DUMP MEMORY IN HEX AND ASCII

\*

551F	BD 43 B5	DUMP	JSR	GETHEX	GET ADDRESS
5522	25 62		BCS	DUMP6	
5524	BD 42 94	DUMP1	JSR	PCRLF	PRINT CR & LF
5527	FE 40 5D		LDX	VALUE	GET ADDRESS
552A	86 10		LDA A	#16	SET COUNT
552C	B7 40 67		STA A	CNTR	
552F	C6 10	DUMP2	LDA B	#16.	SET COUNTER
5531	FF 40 5F		STX	POINT	SAVE POSITION
5534	CE 40 5F		LDX	#POINT	
5537	BD 43 26		JSR	OUTADR	PRINT ADDRESS
553A	BD 49 4F		JSR	OUTSP	PRINT SPACE
553D	FE 40 5F		LDX	POINT	RESET POINTER
5540	37	DUMP25	PSH B		SAVE COUNT
5541	BD 43 29		JSR	OUTHEX	
5544	BD 49 4F		JSR	OUTSP	PRINT DATA & SPACE
5547	08		INX		BUMP TO NEXT
5548	33		PUL B		RESET COUNT
5549	5A		DEC B		DEC THE COUNT
554A	26 F4		BNE	DUMP25	
554C	FE 40 5F		LDX	POINT	RESET POINTER
554F	C6 10		LDA B	#16.	RESET COUNT
5551	A6 00	DUMP27	LDA A	0,X	GET CHARACTER
5553	84 7F		AND A	#\$7F	MASK PARITY
5555	81 1F		CMP A	#\$1F	CONTROL?
5557	22 02		BHI	DUMP28	
5559	86 5F		LDA A	#'_	SET UP ' _ '
555B	BD 41 09	DUMP28	JSR	PUTCHR	OUTPUT CHARACTER
555E	08		INX		BUMP TO NEXT
555F	5A		DEC B		DEC THE COUNT
5560	26 EF		BNE	DUMP27	
5562	BD 42 94		JSR	PCRLF	PRINT CR & LF
5565	7A 40 67		DEC	CNTR	DEC THE COUNT
5568	26 C5		BNE	DUMP2	
556A	7C 40 5D		INC	VALUE ←	BUMP ADDRESS
556D	BD 41 06	DUMP5	JSR	GETCHR	GET RESPONSE
5570	81 0D		CMP A	#\$D	IS IT RETURN?
5572	27 12		BEQ	DUMP6	
5574	84 5F		AND A	#\$5F	MAKE UPPER CASE



5576	81	46		CMP A	#'F	IS IT 'F'?
5578	27	AA		BEQ	DUMP1	
557A	81	42		CMP A	#'B	IS IT 'B'?
557C	26	EF		BNE	DUMP5	
557E	7A	40	5D	DEC	VALUE	DEC THE ADDRESS
5581	7A	40	5D	DEC	VALUE	
5584	20	9E		BRA	DUMP1	
5586	7E	41	95	DUMP6	JMP	EXEC
						RETURN TO EXEC

\*  
\* DO DOS COMMAND  
\*

5589	FE	AC	14	DODOS	LDX	DOSBUF	SAVE DOS POINTER
558C	FF	40	61		STX	POINT2	
558F	FE	40	50		LDX	BUFPT	
5592	FF	AC	14		STX	DOSBUF	SET DOS POINTER
5595	BD	AD	4B		JSR	DOCMND	GO DO COMMAND
5598	FE	40	61		LDX	POINT2	RESET POINTER
559B	FF	AC	14		STX	DOSBUF	
559E	BD	B4	03		JSR	FMSCLS	CLOSE ALL
55A1	20	E3			BRA	DUMP6	

\*  
\* FIND STRING IN MEMORY  
\*

55A3	BD	4C	DD	FIND	JSR	GETRNG	GET RANGE
55A6	FE	40	63		LDX	FIRST	SET START POS
55A9	FF	40	61		STX	POINT2	
55AC	CE	40	00		LDX	#LINBUF	POINT TO BUFFER
55AF	FF	40	56		STX	DATPNT	
55B2	BD	43	94		JSR	PRVW	CHECK NEXT CHAR
55B5	81	22			CMP A	#'"	IS IT "'"?
55B7	26	0F			BNE	FIND3	
55B9	BD	43	61		JSR	NXTCH	USE THE CHAR
55BC	BD	43	61	FIND2	JSR	NXTCH	GET CHARACTER
55BF	81	0D			CMP A	#\$D	IS TI CR?
55C1	27	19			BEQ	FIND4	
55C3	A7	00			STA A	0, X	SAVE CHARACTER
55C5	08				INX		BUMP POINTER
55C6	20	F4			BRA	FIND2	
55C8	BD	43	B5	FIND3	JSR	GETHEX	GET HEX VALUE
55CB	FE	40	56		LDX	DATPNT	
55CE	5D				TST B		ANY HEX?
55CF	27	0B			BEQ	FIND4	
55D1	B6	40	5E		LDA A	VALUE+1	GET VALUE
55D4	A7	00			STA A	0, X	SAVE CHAR
55D6	08				INX		
55D7	FF	40	56		STX	DATPNT	SAVE POSITION
55DA	20	EC			BRA	FIND3	REPEAT
55DC	FF	40	56	FIND4	STX	DATPNT	MARK END
55DF	CE	40	00	FIND5	LDX	#LINBUF	POINT TO BUFFER
55E2	FF	40	5F	FIND6	STX	POINT	SAVE POSITION



55E5 A6 00	LDA A	0,X	GET CHARACTER
55E7 FE 40 63	LDX	FIRST	
55EA BC 40 65	CPX	LAST	FINISHED?
55ED 27 2E	BEQ	FIND8	
55EF A1 00	CMP A	0,X	COMPARE CHARACTERS
55F1 26 1E	BNE	FIND7	
55F3 08	INX		BUMP TO NEXT
55F4 FF 40 63	STX	FIRST	
55F7 FE 40 5F	LDX	POINT	
55FA 08	INX		BUMP TO NEXT HERE TOO
55FB BC 40 56	CPX	DATPNT	END?
55FE 26 E2	BNE	FIND6	
5600 CE 40 61	LDX	#POINT2	FOUND IT!
5603 BD 42 94	JSR	PCRLF	PRINT CR & LF
5606 BD 43 26	JSR	OUTADR	PRINT ADDRESS
5609 FE 40 63	LDX	FIRST	
560C FF 40 61	STX	POINT2	
560F 20 CE	BRA	FIND5	REPEAT
5611 FE 40 61	LDX	POINT2	RESET POINTER
5614 08	INX		
5615 FF 40 63	STX	FIRST	
5618 FF 40 61	STX	POINT2	
561B 20 C2	BRA	FIND5	REPEAT
561D 7E 41 95	JMP	EXEC	GO TO EXEC

\*  
\* FILL MEMORY  
\*

5620 BD 4C DD	FILL	JSR	GETRNG	GET RANGE
5623 BD 43 B5		JSR	GETHEX	GET FILL CHAR
5626 B6 40 5E		LDA A	VALUE+1	
5629 FE 40 63		LDX	FIRST	POINT TO START
562C A7 00	FILL2	STA A	0,X	STORE CHAR
562E BC 40 65		CPX	LAST	FINISHED?
5631 27 EA		BEQ	FIND8	
5633 08		INX		BUMP TO NEXT
5634 20 F6		BRA	FILL2	REPEAT

\*  
\* SET REAL TIME BREAKPOINT  
\*

5636 BD 4C 63	RT	JSR	TSTTRM	CHECK TERM
5639 26 07		BNE	RT2	
563B 7D 40 69		TST	RTBF	IS ONE SET?
563E 27 7D		BEQ	RTB4	
5640 20 66		BRA	RTB2	
5642 BD 43 B5	RT2	JSR	GETHEX	GET ADDRESS
5645 25 79		BCS	RTB6	
5647 A6 00		LDA A	0,X	GET BYTES
5649 B7 40 6C		STA A	RTDAT	SAVE IN TEMPS
564C A6 01		LDA A	1,X	
564E E6 02		LDA B	2,X	



5650 B7 40 6D	STA A	RTDAT+1	
5653 F7 40 6E	STA B	RTDAT+2	
5656 86 7E	LDA A	#\$7E	SET UP 'JUMP'
5658 A7 00	STA A	0, X	
565A 01	NOP		DELAY
565B A1 00	CMP A	0, X	CHECK MEMORY
565D 26 25	BNE	RT4	
565F B7 40 D4	STA A	OP	SET OP TOO
5662 CE 56 87	LDX	#RTB	SET POINTER
5665 FF 40 61	STX	POINT2	
5668 FE 40 5D	LDX	VALUE	RESET X
566B B6 40 61	LDA A	POINT2	SET ADDRESS
566E A7 01	STA A	1, X	
5670 B6 40 62	LDA A	POINT2+1	
5673 A7 02	STA A	2, X	
5675 7C 40 69	INC	RTBF	SET FLAG
5678 FF 40 6A	STX	RTAD	SAVE ADDRESS
567B FE 40 93	LDX	PC	GET PC
567E FF 40 D5	STX	OP+1	SET IN OP
5681 7E 47 02	JMP	TRCOL5	GO EXECUTE
5684 7E 41 C6	RT4 JMP	ILIN	REPORT ERROR

\*  
 \* PROCESS REAL TIME BREAKPOINT  
 \*

5687 36	RTB	PSH A	SAVE A
5688 07		TPA	GET STATUS
5689 B7 40 8C		STA A CC	
568C 32		PUL A	RESTORE A
568D B7 40 8D		STA A AR	SAVE ALL REGISTERS
5690 F7 40 8E		STA B BR	
5693 FF 40 8F		STX XR	
5696 BF 40 91		STS SP	
5699 BE 40 95		LDS SSP	RESET STACK POINTER
569C FE 40 6A		LDX RTAD	GET ADDRESS
569F FF 40 93		STX PC	SAVE AS NEW PC
56A2 FF 40 97		STX OLDPC	
56A5 BD 48 D1		JSR PRREG2	PRINT REGISTERS
56A8 FE 40 6A	RTB2	LDX RTAD	GET ADDRESS
56AB B6 40 6C		LDA A RTDAT	RESET INSTRUCTIONS
56AE A7 00		STA A 0, X	
56B0 B6 40 6D		LDA A RTDAT+1	
56B3 A7 01		STA A 1, X	
56B5 B6 40 6E		LDA A RTDAT+2	
56B8 A7 02		STA A 2, X	
56BA 7F 40 69		CLR RTBF	CLEAR FLAG
56BD 7E 41 95	RTB4	JMP EXEC	RETURN TO EXEC
56C0 7E 41 CE	RTB6	JMP SYNER	REPORT ERROR

\*  
 \* CHECK SP PROTECTION  
 \*



56C3	7D	40	9B	CHKSP	TST	MODE	CHECK MODE
56C6	27	34			BEQ	CHKSP4	
56C8	4F				CLR A		
56C9	5D				TST B		OFFSET NEGATIVE?
56CA	2A	01			BPL	CHKSP1	
56CC	43				COM A		SIGN EXTEND
56CD	FB	40	92	CHKSP1	ADD B	SP+1	ADD TO SP
56D0	B9	40	91		ADC A	SP	
56D3	F7	40	84		STA B	TSP+1	SAVE RESULT
56D6	B7	40	83		STA A	TSP	
56D9	B1	40	81		CMP A	MAXSP	CHECK IF MAX SP
56DC	22	0D			BHI	CHKSP3	
56DE	25	05			BLO	CHKSP2	
56E0	F1	40	82		CMP B	MAXSP+1	CHECK LSB
56E3	24	06			BHS	CHKSP3	
56E5	B7	40	81	CHKSP2	STA A	MAXSP	SAVE NEW MAX
56E8	F7	40	82		STA B	MAXSP+1	
56EB	BD	4B	6E	CHKSP3	JSR	MPR	CHECK PROTECTION
56EE	26	0C			BNE	CHKSP4	
56F0	FE	40	97		LDX	OLDPC	RESET PC
56F3	FF	40	93		STX	PC	
56F6	CE	5A	38		LDX	#MSPST	POINT TO STRING
56F9	7E	4B	34		JMP	PRCH3	PRINT IT
56FC	39			CHKSP4	RTS		RETURN

\*  
\* PRINT STACK DEPTH  
\*

56FD	BD	42	94	DEPTH	JSR	PCRLF	PRINT CR & LF
5700	CE	40	81		LDX	#MAXSP	POINT TO MAX
5703	BD	43	26		JSR	OUTADR	PRINT IT
5706	20	11			BRA	FLAG5	

\*  
\* SET FLAG LOCATION  
\*

5708	7F	40	A0	FLAG	CLR	FLAGB	
570B	BD	4C	63		JSR	TSTTRM	CHECK TERM
570E	27	09			BEQ	FLAG5	
5710	BD	4C	37		JSR	HEQVAL	GET ADDRESS
5713	FF	40	9E		STX	FLAGA	SAVE ADDRESS
5716	7C	40	A0		INC	FLAGB	SET FLAG BYTE
5719	7E	41	95	FLAG5	JMP	EXEC	RETURN

\*  
\* PRINT SINGLE DECIMAL  
\*

571C	B7	40	5E	SINDEC	STA A	VALUE+1	SAVE VALUE
571F	7F	40	5D		CLR	VALUE	
5722	5F				CLR B		CLEAR FLAG
5723	CE	40	5D		LDX	#VALUE	POINT TO VALUE



5726 7E 42 D5 JMP OUTDEC PRINT NUMBER

\*  
\* PRINT MACHINE CONFIGURATION  
\*

5729 BD 42 94	MACH	JSR	PCRLF	PRINT CR & LF
572C CE 59 64		LDX	#XMM	POINT TO M STR
572F 8D 27		BSR	MACH2	PRINT IT
5731 B6 40 9B		LDA A	MODE	GET MODE
5734 8D E6		BSR	SINDEC	PRINT IT
5736 CE 59 67		LDX	#XMT	PRINT TRACE VALUE
5739 8D 1D		BSR	MACH2	
573B B6 40 9D		LDA A	TRCF	
573E 8D DC		BSR	SINDEC	
5740 CE 59 58		LDX	#NST	PRINT NEST VALUE
5743 8D 13		BSR	MACH2	
5745 B6 40 A5		LDA A	NSTRP	
5748 8D D2		BSR	SINDEC	
574A CE 59 5C		LDX	#XVST	PRINT INSTRUCTION COUNT
574D 8D 09		BSR	MACH2	
574F CE 40 77		LDX	#MAXC	
5752 5F		CLR B		
5753 BD 42 D5		JSR	OUTDEC	
5756 20 03		BRA	MACH4	
5758 7E 42 87	MACH2	JMP	PDATA1	PRINT STRING
575B CE 59 50	MACH4	LDX	#SPST	PRINT STOP ADDRESS
575E 8D F8		BSR	MACH2	
5760 CE 40 73		LDX	#END	
5763 BD 43 26		JSR	OUTADR	
5766 CE 59 6B		LDX	#XMIT	PRINT I TRAP STATE
5769 8D ED		BSR	MACH2	
576B B6 40 AB		LDA A	ITRF	
576E 8D 24		BSR	PRONF	PRINT ON/OFF
5770 CE 59 70		LDX	#XMXT	PRINT XFR TRAP STATE
5773 8D E3		BSR	MACH2	
5775 B6 40 A6		LDA A	XFRT	
5778 8D 1A		BSR	PRONF	PRINT ON/OFF
577A CE 59 75		LDX	#XMIRO	PRINT IRQ COUNT
577D 8D D9		BSR	MACH2	
577F CE 40 7D		LDX	#IRQC	
5782 5F		CLR B		
5783 BD 42 D5		JSR	OUTDEC	
5786 CE 59 7B		LDX	#XMNMI	PRINT NMI COUNT
5789 8D CD		BSR	MACH2	
578B CE 40 79		LDX	#NMIC	
578E 5F		CLR B		
578F BD 42 D5		JSR	OUTDEC	
5792 20 85	MACH6	BRA	FLAG5	RETURN TO EXEC

\*  
\* PRINT ON OR OFF STATUS  
\*



5794 27 05	PRONF	BEQ	PRONF2	
5796 CE 59 81		LDX	#XONST	PRINT ON
5799 20 BD		BRA	MACH2	
579B CE 59 84	PRONF2	LDX	#XOFST	PRINT OFF
579E 20 B8		BRA	MACH2	

\*  
\* CLEAR MESSAGE TABLE  
\*

57A0 CE 3F 80	CLM	LDX	#MSGTB	POINT TO TABLE
57A3 FF 40 B6		STX	NXTMSG	SAVE POINTER
57A6 20 EA		BRA	MACH6	

\* COMMAND TABLE

57A8	COMTBL	EQU	*
57A8 41		FCC	'ASM'
57AB 00		FCB	0
57AC 53 5F		FDB	DORASM
57AE 42		FCC	'B'
57AF 00		FCB	0
57B0 4F B7		FDB	BRK
57B2 42		FCC	'BOUNDS'
57B8 00		FCB	0
57B9 4D 85		FDB	BOUNDS
57BB 42		FCC	'BP'
57BD 00		FCB	0
57BE 52 05		FDB	PBP
57C0 43		FCC	'CALC'
57C4 00		FCB	0
57C5 52 BE		FDB	CALC
57C7 43		FCC	'CLB'
57CA 00		FCB	0
57CB 52 2D		FDB	CLB
57CD 43		FCC	'CLH'
57D0 00		FCB	0
57D1 52 49		FDB	CLH
57D3 43		FCC	'CLM'
57D6 00		FCB	0
57D7 57 A0		FDB	CLM
57D9 43		FCC	'CLP'
57DC 00		FCB	0
57DD 54 D5		FDB	CLP
57DF 44		FCC	'DIS'
57E2 00		FCB	0
57E3 4E 8E		FDB	DIS
57E5 44		FCC	'DEPTH'
57EA 00		FCB	0
57EB 56 FD		FDB	DEPTH
57ED 44		FCC	'DELAY'
57F2 00		FCB	0
57F3 52 8C		FDB	SDELY
57F5 44		FCC	'DUMP'



57F9 00	FCB	0
57FA 55 1F	FDB	DUMP
57FC 45	FCC	'EXIT'
5800 00	FCB	0
5801 41 0C	FDB	WARMS
5803 46	FCC	'FIND'
5807 00	FCB	0
5808 55 A3	FDB	FIND
580A 46	FCC	'FILL'
580E 00	FCB	0
580F 56 20	FDB	FILL
5811 46	FCC	'FLAG'
5815 00	FCB	0
5816 57 08	FDB	FLAG
5818 47	FCC	'GO'
581A 00	FCB	0
581B 44 3F	FDB	GO
581D 48	FCC	'HIST'
5821 00	FCB	0
5822 51 7F	FDB	HIST
5824 49	FCC	'IND'
5827 00	FCB	0
5828 52 97	FDB	SIND
582A 49	FCC	'INST'
582E 00	FCB	0
582F 4C 41	FDB	INST
5831 49	FCC	'IRQ'
5834 00	FCB	0
5835 52 A8	FDB	SIRQ
5837 49	FCC	'ITRAP'
583C 00	FCB	0
583D 54 C3	FDB	ITRAP
583F 4A	FCC	'JUMP'
5843 00	FCB	0
5844 44 46	FDB	JUMP
5846 4D	FCC	'MEM'
5849 00	FCB	0
584A 4E DE	FDB	XMEM
584C 4D	FCC	'MACH'
5850 00	FCB	0
5851 57 29	FDB	MACH
5853 4D	FCC	'MODE'
5857 00	FCB	0
5858 4D 72	FDB	CMODE
585A 4E	FCC	'NEST'
585E 00	FCB	0
585F 4C 53	FDB	NEST
5861 4E	FCC	'NMI'
5864 00	FCB	0
5865 52 B3	FDB	SNMI
5867 50	FCC	'PROT'
586B 00	FCB	0
586C 4C F9	FDB	PROT
586E 50	FCC	'PAST'



5872 00	FCB	0
5873 4C 8C	FDB	PAST
5875 52	FCC	'REG'
5878 00	FCB	0
5879 48 BC	FDB	DREG
587B 52	FCC	'RESET'
5880 00	FCB	0
5881 41 1B	FDB	COLD
5883 52	FCC	'RET'
5886 00	FCB	0
5887 4E 5C	FDB	RTRN
5889 52	FCC	'RT'
588B 00	FCB	0
588C 56 36	FDB	RT
588E 53	FCC	'SET'
5891 00	FCB	0
5892 4F 48	FDB	SET
5894 53	FCC	'SIM'
5897 00	FCB	0
5898 52 5E	FDB	CSIM
589A 53	FCC	'START'
589F 00	FCB	0
58A0 44 71	FDB	SIMUL
58A2 53	FCC	'STACK'
58A7 00	FCB	0
58A8 53 3B	FDB	PSTK
58AA 53	FCC	'STATES'
58B0 00	FCB	0
58B1 52 7A	FDB	PSTAT
58B3 53	FCC	'STEP'
58B7 00	FCB	0
58B8 4B E6	FDB	SSM
58BA 53	FCC	'STOP'
58BE 00	FCB	0
58BF 4E CD	FDB	SSTOP
58C1 54	FCC	'TRACE'
58C6 00	FCB	0
58C7 4E C2	FDB	STRAC
58C9 54	FCC	'TRAIL'
58CE 00	FCB	0
58CF 4C 7E	FDB	TRAIL
58D1 54	FCC	'TSIM'
58D5 00	FCB	0
58D6 52 57	FDB	TCSIM
58D8 58	FCC	'X'
58D9 00	FCB	0
58DA 55 89	FDB	DODOS
58DC 58	FCC	'XFR'
58DF 00	FCB	0
58E0 4E 40	FDB	STXFR
58E2 00	FCB	0

END TABLE

\* DECIMAL CONSTANT TABLE



58E3 27 10	CONTBL	FDB	10000	CONSTANTS
58E5 03 E8		FDB	1000	
58E7 00 64		FDB	100	
58E9 00 0A		FDB	10	

## \* FUNCTION TABLE

58EB 47 30	FUNTAB	FDB	PBSR
58ED 47 8A		FDB	PJMP
58EF 47 7D		FDB	PJSR
58F1 47 B0		FDB	PSUBR
58F3 48 51		FDB	PRTI
58F5 48 10		FDB	PSWI
58F7 48 49		FDB	PWAI

## \* REGISTER POINTER TABLE

58F9 40 8D	RGTAB	FDB	AR
58FB 40 8E		FDB	BR
58FD 40 8C		FDB	CC
58FF 40 99		FDB	NESTC
5901 40 91		FDB	SP
5903 40 8F		FDB	XR
5905 40 93		FDB	PC

\*

## \* REGISTER TABLE

\*

5907 41	RGT	FCB	'A, 1
5909 40 8D		FDB	AR
590B 42		FCB	'B, 1
590D 40 8E		FDB	BR
590F 43		FCB	'C, 1
5911 40 8C		FDB	CC
5913 4E		FCB	'N, 1
5915 40 99		FDB	NESTC
5917 53		FCB	'S, 2
5919 40 91		FDB	SP
591B 58		FCB	'X, 2
591D 40 8F		FDB	XR
591F 50		FCB	'P, 2
5921 40 93		FDB	PC
5923 00		FCB	0

\*

## \* TYPE TABLE

\*

5924 53	TYPTB	FCB	'S, 1, 'R, 2, 'T, 4, 'U, 8
592C 5A		FCB	'Z, \$10, 'H, \$20, 'M, \$40, 'J, \$80
5934 00		FCB	0

## \* STRINGS



5935 2A	PRMPT	FCC	'**'
5937 04		FCB	4
5938 20	CLCP	FCC	'='
593A 04		FCB	4
593B 57	WHATST	FCC	'WHAT?'
5940 04		FCB	4
5941 43	CCST	FCC	'C='
5943 04		FCB	4
5944 20	AST	FCC	'A='
5947 04		FCB	4
5948 20	BST	FCC	'B='
594B 04		FCB	4
594C 20	XST	FCC	'X='
594F 04		FCB	4
5950 20	SPST	FCC	'S='
5953 04		FCB	4
5954 20	PCST	FCC	'P='
5957 04		FCB	4
5958 20	NST	FCC	'N='
595B 04		FCB	4
595C 20	XVST	FCC	'I='
595F 04		FCB	4
5960 20	FMST	FCC	'F='
5963 04		FCB	4
5964 4D	XMM	FCC	'M='
5966 04		FCB	4
5967 20	XMT	FCC	'T='
596A 04		FCB	4
596B 20	XMIT	FCC	'IT='
596F 04		FCB	4
5970 20	XMXT	FCC	'XT='
5974 04		FCB	4
5975 20	XMIRQ	FCC	'IRQ='
597A 04		FCB	4
597B 20	XNMNI	FCC	'NMI='
5980 04		FCB	4
5981 4F	XONST	FCC	'ON'
5983 04		FCB	4
5984 4F	XOFST	FCC	'OFF'
5987 04		FCB	4
5988 20	INPST	FCC	'IN'
598C 04		FCB	4
598D 20	OUTST	FCC	'OUT'
5992 04		FCB	4
5993 2C	IXST	FCC	'X'
5995 04		FCB	4
5996 3F	DELST	FCC	'??'
5998 04		FCB	4
5999 0D 0A	CRLFST	FDB	\$D0A, 0, 4
599F 53	STST	FCC	'STATES = '
59A8 04		FCB	4
59A9 22	STPST	FCC	'"STOP" AT '
59B3 04		FCB	4



59B4 49	CTST	FCC	'IC TIMEOUT AT '
59C2 04		FCB	4
59C3 49	ILOPST	FCC	'ILLEGAL OP CODE AT '
59D5 04		FCB	4
59D6 49	ITRST	FCC	'I TRAP AT '
59E0 04		FCB	4
59E1 4C	TRST	FCC	'LAST XFR FROM '
59EF 04		FCB	4
59F0 53	SYNST	FCC	'SYNTAX ERROR'
59FC 04		FCB	4
59FD 45	EXPST	FCC	'EP TRAP AT '
5A08 04		FCB	4
5A09 57	WPST	FCC	'WP TRAP AT '
5A14 04		FCB	4
5A15 45	MMP1ST	FCC	'EX - MP TRAP AT '
5A25 04		FCB	4
5A26 52	MMP2ST	FCC	'REF - MP TRAP AT '
5A37 04		FCB	4
5A38 53	MSPST	FCC	'SP TRAP AT '
5A43 04		FCB	4
5A44 54	TBLOV	FCC	'TABLE OVERFLOW'
5A52 04		FCB	4
5A53 57	WPRST	FCC	'WRITE'
5A58 04		FCB	4
5A59 45	XPST	FCC	'EXECUTE'
5A60 04		FCB	4
5A61 4D	MPST	FCC	'MEMORY'
5A67 04		FCB	4
5A68 53	SMPST	FCC	'SIMULATE'
5A70 04		FCB	4
5A71 20	PRTSTR	FCC	' PROTECTION'
5A7C 04		FCB	4
5A7D 4E	NSTST	FCC	'NC TRAP AT '
5A88 04		FCB	4
5A89 52	NERST	FCC	'RTS IN LEVEL 0 AT '
5A9B 04		FCB	4
5A9C 52	RTADS	FCC	'RETURN = '
5AA5 04		FCB	4
5AA6 4E	NORTS	FCC	'NEST LEVEL IS 0'
5AB5 04		FCB	4
5AB6 58	XFRST	FCC	'XFR TRAP AT '
5AC2 04		FCB	4
5AC3 4F	BOVST	FCC	'OVERFLOW'
5ACB 04		FCB	4
5ACC 4D	HMONS	FCC	'MON XFR AT '
5AD7 04		FCB	4
5AD8 4F	OPHST	FCC	'OP HALT AT '
5AE3 04		FCB	4
5AE4 20	DSHST	FCC	' - '
5AE7 04		FCB	4

\*  
\* MNEMONIC AND INFO TABLE  
\*

SAES

INIT	LDA A	# \$03
	STA A	\$0004
	LDA A	# \$15
	STA A	\$0404
	JMP	\$3100
	PSH B	
	LDA B	\$0004
	LSR B	
	BCC	INCH1
	LDA A	\$0005

SAES

INCH  
INCH1

	BRA	X+3
OUTCH	PUL B	
OUTCH1	LDA B	\$0004
	ASR B	
	ASR B	
	BCC	OUTCH1
	STO A	\$0005
	PUL B	
	RTS	



5B00		ORG	\$5B00
5B00 2A	OPTAB	FCC	'****'
5B03 00		FCB	00
5B04 4E		FCC	'NOP'
5B07 12		FCB	\$12
5B08 2A		FCC	'****'
5B0B 00		FCB	00
5B0C 2A		FCC	'****'
5B0F 00		FCB	00
5B10 2A		FCC	'****'
5B13 00		FCB	00
5B14 2A		FCC	'****'
5B17 00		FCB	00
5B18 54		FCC	'TAP'
5B1B 12		FCB	\$12
5B1C 54		FCC	'TPA'
5B1F 12		FCB	\$12
5B20 49		FCC	'INX'
5B23 14		FCB	\$14
5B24 44		FCC	'DEX'
5B27 14		FCB	\$14
5B28 43		FCC	'CLV'
5B2B 12		FCB	\$12
5B2C 53		FCC	'SEV'
5B2F 12		FCB	\$12
5B30 43		FCC	'CLC'
5B33 12		FCB	\$12
5B34 53		FCC	'SEC'
5B37 12		FCB	\$12
5B38 43		FCC	'CLI'
5B3B 12		FCB	\$12
5B3C 53		FCC	'SEI'
5B3F 12		FCB	\$12
5B40 53		FCC	'SBA' \$10
5B43 12		FCB	\$12
5B44 43		FCC	'CBA'
5B47 12		FCB	\$12
5B48 2A		FCC	'****'
5B4B 00		FCB	00
5B4C 2A		FCC	'****'
5B4F 00		FCB	00
5B50 2A		FCC	'****'
5B53 00		FCB	00
5B54 2A		FCC	'****'
5B57 00		FCB	00
5B58 54		FCC	'TAB'
5B5B 12		FCB	\$12
5B5C 54		FCC	'TBA'
5B5F 12		FCB	\$12
5B60 2A		FCC	'****'
5B63 00		FCB	00
5B64 44		FCC	'DAA'



5B67 12	FCB	\$12	
5B68 2A	FCC	'****'	
5B6B 00	FCB	00	
5B6C 41	FCC	'ABA'	
5B6F 12	FCB	\$12	
5B70 2A	FCC	'****'	
5B73 00	FCB	00	
5B74 2A	FCC	'****'	
5B77 00	FCB	00	
5B78 2A	FCC	'****'	
5B7B 00	FCB	00	
5B7C 2A	FCC	'****'	
5B7F 00	FCB	00	
5B80 42	FCC	'BRA'	\$20
5B83 24	FCB	\$24	
5B84 2A	FCC	'****'	
5B87 00	FCB	00	
5B88 42	FCC	'BHI'	
5B8B 24	FCB	\$24	
5B8C 42	FCC	'BLS'	
5B8F 24	FCB	\$24	
5B90 42	FCC	'BCC'	
5B93 24	FCB	\$24	
5B94 42	FCC	'BCS'	
5B97 24	FCB	\$24	
5B98 42	FCC	'BNE'	
5B9B 24	FCB	\$24	
5B9C 42	FCC	'BEQ'	
5B9F 24	FCB	\$24	
5BA0 42	FCC	'BVC'	
5BA3 24	FCB	\$24	
5BA4 42	FCC	'BVS'	
5BA7 24	FCB	\$24	
5BA8 42	FCC	'BPL'	
5BAB 24	FCB	\$24	
5BAC 42	FCC	'BMI'	
5BAF 24	FCB	\$24	
5BB0 42	FCC	'BGE'	
5BB3 24	FCB	\$24	
5BB4 42	FCC	'BLT'	
5BB7 24	FCB	\$24	
5BB8 42	FCC	'BGT'	
5BBB 24	FCB	\$24	
5BBC 42	FCC	'BLE'	
5BBF 24	FCB	\$24	
5BC0 54	FCC	'TSX'	\$30
5BC3 14	FCB	\$14	
5BC4 49	FCC	'INS'	
5BC7 14	FCB	\$14	
5BC8 50	FCC	'PUL'	
5BCB 14	FCB	\$14	
5BCC 50	FCC	'PUL'	
5BCF 14	FCB	\$14	
5BD0 44	FCC	'DES'	



5BD3 14	FCB	\$14	
5BD4 54	FCC	'TXS'	
5BD7 14	FCB	\$14	
5BD8 50	FCC	'PSH'	
5BDB 14	FCB	\$14	
5BDC 50	FCC	'PSH'	
5BDF 14	FCB	\$14	
5BE0 2A	FCC	'***'	
5BE3 00	FCB	00	
5BE4 52	FCC	'RTS'	
5BE7 15	FCB	\$15	
5BE8 2A	FCC	'***'	
5BEB 00	FCB	00	
5BEC 52	FCC	'RTI'	
5BEF 1A	FCB	\$1A	
5BF0 2A	FCC	'***'	
5BF3 00	FCB	00	
5BF4 2A	FCC	'***'	
5BF7 00	FCB	00	
5BF8 57	FCC	'WAI'	
5BFB 19	FCB	\$19	
5BFC 53	FCC	'SWI'	
5BFF 1C	FCB	\$1C	
5C00 4E	FCC	'NEG'	\$40
5C03 12	FCB	\$12	
5C04 2A	FCC	'***'	
5C07 00	FCB	00	
5C08 2A	FCC	'***'	
5C0B 00	FCB	00	
5C0C 43	FCC	'COM'	
5C0F 12	FCB	\$12	
5C10 4C	FCC	'LSR'	
5C13 12	FCB	\$12	
5C14 2A	FCC	'***'	
5C17 00	FCB	00	
5C18 52	FCC	'ROR'	
5C1B 12	FCB	\$12	
5C1C 41	FCC	'ASR'	
5C1F 12	FCB	\$12	
5C20 41	FCC	'ASL'	
5C23 12	FCB	\$12	
5C24 52	FCC	'ROL'	
5C27 12	FCB	\$12	
5C28 44	FCC	'DEC'	
5C2B 12	FCB	\$12	
5C2C 2A	FCC	'***'	
5C2F 00	FCB	00	
5C30 49	FCC	'INC'	
5C33 12	FCB	\$12	
5C34 54	FCC	'TST'	
5C37 12	FCB	\$12	
5C38 2A	FCC	'***'	
5C3B 00	FCB	00	
5C3C 43	FCC	'CLR'	



5C3F 12	FCB	\$12	012	000	01	0000
5C40 4E	FCC	'NEG'	\$50	000	02	0000
5C43 12	FCB	\$12	012	000	03	0000
5C44 2A	FCC	'***'	000	000	04	0000
5C47 00	FCB	00	000	000	05	0000
5C48 2A	FCC	'***'	000	000	06	0000
5C4B 00	FCB	00	000	000	07	0000
5C4C 43	FCC	'COM'	000	000	08	0000
5C4F 12	FCB	\$12	000	000	09	0000
5C50 4C	FCC	'LSR'	000	000	10	0000
5C53 12	FCB	\$12	000	000	11	0000
5C54 2A	FCC	'***'	000	000	12	0000
5C57 00	FCB	00	000	000	13	0000
5C58 52	FCC	'ROR'	000	000	14	0000
5C5B 12	FCB	\$12	000	000	15	0000
5C5C 41	FCC	'ASR'	000	000	16	0000
5C5F 12	FCB	\$12	000	000	17	0000
5C60 41	FCC	'ASL'	000	000	18	0000
5C63 12	FCB	\$12	000	000	19	0000
5C64 52	FCC	'ROL'	000	000	20	0000
5C67 12	FCB	\$12	000	000	21	0000
5C68 44	FCC	'DEC'	000	000	22	0000
5C6B 12	FCB	\$12	000	000	23	0000
5C6C 2A	FCC	'***'	000	000	24	0000
5C6F 00	FCB	00	000	000	25	0000
5C70 49	FCC	'INC'	000	000	26	0000
5C73 12	FCB	\$12	000	000	27	0000
5C74 54	FCC	'TST'	000	000	28	0000
5C77 12	FCB	\$12	000	000	29	0000
5C78 2A	FCC	'***'	000	000	30	0000
5C7B 00	FCB	00	000	000	31	0000
5C7C 43	FCC	'CLR'	000	000	32	0000
5C7F 12	FCB	\$12	000	000	33	0000
5C80 4E	FCC	'NEG'	\$60	000	34	0000
5C83 27	FCB	\$27	000	000	35	0000
5C84 2A	FCC	'***'	000	000	36	0000
5C87 00	FCB	00	000	000	37	0000
5C88 2A	FCC	'***'	000	000	38	0000
5C8B 00	FCB	00	000	000	39	0000
5C8C 43	FCC	'COM'	000	000	40	0000
5C8F 27	FCB	\$27	000	000	41	0000
5C90 4C	FCC	'LSR'	000	000	42	0000
5C93 27	FCB	\$27	000	000	43	0000
5C94 2A	FCC	'***'	000	000	44	0000
5C97 00	FCB	00	000	000	45	0000
5C98 52	FCC	'ROR'	000	000	46	0000
5C9B 27	FCB	\$27	000	000	47	0000
5C9C 41	FCC	'ASR'	000	000	48	0000
5C9F 27	FCB	\$27	000	000	49	0000
5CA0 41	FCC	'ASL'	000	000	50	0000
5CA3 27	FCB	\$27	000	000	51	0000
5CA4 52	FCC	'ROL'	000	000	52	0000
5CA7 27	FCB	\$27	000	000	53	0000
5CA8 44	FCC	'DEC'	000	000	54	0000



5CAB 27	FCB	\$27	
5CAC 2A	FCC	'***'	
5CAF 00	FCB	00	
5CB0 49	FCC	'INC'	
5CB3 27	FCB	\$27	
5CB4 54	FCC	'TST'	
5CB7 27	FCB	\$27	
5CB8 4A	FCC	'JMP'	
5CBB 24	FCB	\$24	
5CBC 43	FCC	'CLR'	
5CBF 27	FCB	\$27	
5CC0 4E	FCC	'NEG'	\$70
5CC3 36	FCB	\$36	
5CC4 2A	FCC	'***'	
5CC7 00	FCB	00	
5CC8 2A	FCC	'***'	
5CCB 00	FCB	00	
5CCC 43	FCC	'COM'	
5CCF 36	FCB	\$36	
5CD0 4C	FCC	'LSR'	
5CD3 36	FCB	\$36	
5CD4 2A	FCC	'***'	
5CD7 00	FCB	00	
5CD8 52	FCC	'ROR'	
5CDB 36	FCB	\$36	
5CDC 41	FCC	'ASR'	
5CDF 36	FCB	\$36	
5CE0 41	FCC	'ASL'	
5CE3 36	FCB	\$36	
5CE4 52	FCC	'ROL'	
5CE7 36	FCB	\$36	
5CE8 44	FCC	'DEC'	
5CEB 36	FCB	\$36	
5CEC 2A	FCC	'***'	
5CEF 00	FCB	00	
5CF0 49	FCC	'INC'	
5CF3 36	FCB	\$36	
5CF4 54	FCC	'TST'	
5CF7 36	FCB	\$36	
5CF8 4A	FCC	'JMP'	
5CFB 33	FCB	\$33	
5CFC 43	FCC	'CLR'	
5CFF 36	FCB	\$36	
5D00 53	FCC	'SUB'	\$80
5D03 22	FCB	\$22	
5D04 43	FCC	'CMP'	
5D07 22	FCB	\$22	
5D08 53	FCC	'SBC'	
5D0B 22	FCB	\$22	
5D0C 2A	FCC	'***'	
5D0F 00	FCB	00	
5D10 41	FCC	'AND'	
5D13 22	FCB	\$22	
5D14 42	FCC	'BIT'	



5D17 22	FCB	\$22	
5D18 4C	FCC	'LDA'	
5D1B 22	FCB	\$22	
5D1C 2A	FCC	'***'	
5D1F 00	FCB	00	
5D20 45	FCC	'EOR'	
5D23 22	FCB	\$22	
5D24 41	FCC	'ADC'	
5D27 22	FCB	\$22	
5D28 4F	FCC	'ORA'	
5D2B 22	FCB	\$22	
5D2C 41	FCC	'ADD'	
5D2F 22	FCB	\$22	
5D30 43	FCC	'CPX'	
5D33 33	FCB	\$33	
5D34 42	FCC	'BSR'	
5D37 28	FCB	\$28	
5D38 4C	FCC	'LDS'	
5D3B 33	FCB	\$33	
5D3C 2A	FCC	'***'	
5D3F 00	FCB	00	
5D40 53	FCC	'SUB'	\$90
5D43 23	FCB	\$23	
5D44 43	FCC	'CMP'	
5D47 23	FCB	\$23	
5D48 53	FCC	'SBC'	
5D4B 23	FCB	\$23	
5D4C 2A	FCC	'***'	
5D4F 00	FCB	00	
5D50 41	FCC	'AND'	
5D53 23	FCB	\$23	
5D54 42	FCC	'BIT'	
5D57 23	FCB	\$23	
5D58 4C	FCC	'LDA'	
5D5B 23	FCB	\$23	
5D5C 53	FCC	'STA'	
5D5F 24	FCB	\$24	
5D60 45	FCC	'EOR'	
5D63 23	FCB	\$23	
5D64 41	FCC	'ADC'	
5D67 23	FCB	\$23	
5D68 4F	FCC	'ORA'	
5D6B 23	FCB	\$23	
5D6C 41	FCC	'ADD'	
5D6F 23	FCB	\$23	
5D70 43	FCC	'CPX'	
5D73 24	FCB	\$24	
5D74 2A	FCC	'***'	
5D77 00	FCB	00	
5D78 4C	FCC	'LDS'	
5D7B 24	FCB	\$24	
5D7C 53	FCC	'STS'	
5D7F 25	FCB	\$25	
5D80 53	FCC	'SUB'	\$A0



5D83 25	FCB	\$25
5D84 43	FCC	'CMP'
5D87 25	FCB	\$25
5D88 53	FCC	'SBC'
5D8B 25	FCB	\$25
5D8C 2A	FCC	'****'
5D8F 00	FCB	00
5D90 41	FCC	'AND'
5D93 25	FCB	\$25
5D94 42	FCC	'BIT'
5D97 25	FCB	\$25
5D98 4C	FCC	'LDA'
5D9B 25	FCB	\$25
5D9C 53	FCC	'STA'
5D9F 26	FCB	\$26
5DA0 45	FCC	'EOR'
5DA3 25	FCB	\$25
5DA4 41	FCC	'ADC'
5DA7 25	FCB	\$25
5DA8 4F	FCC	'ORA'
5DAB 25	FCB	\$25
5DAC 41	FCC	'ADD'
5DAF 25	FCB	\$25
5DB0 43	FCC	'CPX'
5DB3 26	FCB	\$26
5DB4 4A	FCC	'JSR'
5DB7 28	FCB	\$28
5DB8 4C	FCC	'LDS'
5DBB 26	FCB	\$26
5DBC 53	FCC	'STS'
5DBF 27	FCB	\$27
5DC0 53	FCC	'SUB' \$B0
5DC3 34	FCB	\$34
5DC4 43	FCC	'CMP'
5DC7 34	FCB	\$34
5DC8 53	FCC	'SBC'
5DCB 34	FCB	\$34
5DCC 2A	FCC	'****'
5DCF 00	FCB	00
5DD0 41	FCC	'AND'
5DD3 34	FCB	\$34
5DD4 42	FCC	'BIT'
5DD7 34	FCB	\$34
5DD8 4C	FCC	'LDA'
5ddb 34	FCB	\$34
5DDC 53	FCC	'STA'
5DDF 35	FCB	\$35
5DE0 45	FCC	'EOR'
5DE3 34	FCB	\$34
5DE4 41	FCC	'ADC'
5DE7 34	FCB	\$34
5DE8 4F	FCC	'ORA'
5DEB 34	FCB	\$34
5DEC 41	FCC	'ADD'



5DEF 34	FCB	\$34
5DF0 43	FCC	'CPX'
5DF3 35	FCB	\$35
5DF4 4A	FCC	'JSR'
5DF7 39	FCB	\$39
5DF8 4C	FCC	'LDS'
5DFB 35	FCB	\$35
5DFC 53	FCC	'STS'
5DFF 36	FCB	\$36
5E00 53	FCC	'SUB' \$C0
5E03 22	FCB	\$22
5E04 43	FCC	'CMP'
5E07 22	FCB	\$22
5E08 53	FCC	'SBC'
5E0B 22	FCB	\$22
5E0C 2A	FCC	'****'
5E0F 00	FCB	00
5E10 41	FCC	'AND'
5E13 22	FCB	\$22
5E14 42	FCC	'BIT'
5E17 22	FCB	\$22
5E18 4C	FCC	'LDA'
5E1B 22	FCB	\$22
5E1C 2A	FCC	'****'
5E1F 00	FCB	00
5E20 45	FCC	'EOR'
5E23 22	FCB	\$22
5E24 41	FCC	'ADC'
5E27 22	FCB	\$22
5E28 4F	FCC	'ORA'
5E2B 22	FCB	\$22
5E2C 41	FCC	'ADD'
5E2F 22	FCB	\$22
5E30 2A	FCC	'****'
5E33 00	FCB	00
5E34 2A	FCC	'****'
5E37 00	FCB	00
5E38 4C	FCC	'LDX'
5E3B 33	FCB	\$33
5E3C 2A	FCC	'****'
5E3F 00	FCB	00
5E40 53	FCC	'SUB' \$D0
5E43 23	FCB	\$23
5E44 43	FCC	'CMP'
5E47 23	FCB	\$23
5E48 53	FCC	'SBC'
5E4B 23	FCB	\$23
5E4C 2A	FCC	'****'
5E4F 00	FCB	00
5E50 41	FCC	'AND'
5E53 23	FCB	\$23
5E54 42	FCC	'BIT'
5E57 23	FCB	\$23
5E58 4C	FCC	'LDA'



5E5B 23	FCB	\$23	
5E5C 53	FCC	'STA'	
5E5F 24	FCB	\$24	
5E60 45	FCC	'EOR'	
5E63 23	FCB	\$23	
5E64 41	FCC	'ADC'	
5E67 23	FCB	\$23	
5E68 4F	FCC	'ORA'	
5E6B 23	FCB	\$23	
5E6C 41	FCC	'ADD'	
5E6F 23	FCB	\$23	
5E70 2A	FCC	'***'	
5E73 00	FCB	00	
5E74 2A	FCC	'***'	
5E77 00	FCB	00	
5E78 4C	FCC	'LDX'	
5E7B 24	FCB	\$24	
5E7C 53	FCC	'STX'	
5E7F 25	FCB	\$25	
5E80 53	FCC	'SUB' \$E0	
5E83 25	FCB	\$25	
5E84 43	FCC	'CMP'	
5E87 25	FCB	\$25	
5E88 53	FCC	'SBC'	
5E8B 25	FCB	\$25	
5E8C 2A	FCC	'***'	
5E8F 00	FCB	00	
5E90 41	FCC	'AND'	
5E93 25	FCB	\$25	
5E94 42	FCC	'BIT'	
5E97 25	FCB	\$25	
5E98 4C	FCC	'LDA'	
5E9B 25	FCB	\$25	
5E9C 53	FCC	'STA'	
5E9F 26	FCB	\$26	
5EA0 45	FCC	'EOR'	
5EA3 25	FCB	\$25	
5EA4 41	FCC	'ADC'	
5EA7 25	FCB	\$25	
5EA8 4F	FCC	'ORA'	
5EAB 25	FCB	\$25	
5EAC 41	FCC	'ADD'	
5EAF 25	FCB	\$25	
5EB0 2A	FCC	'***'	
5EB3 00	FCB	00	
5EB4 2A	FCC	'***'	
5EB7 00	FCB	00	
5EB8 4C	FCC	'LDX'	
5EBB 26	FCB	\$26	
5EBC 53	FCC	'STX'	
5EBF 27	FCB	\$27	
5EC0 53	FCC	'SUB' \$F0	
5EC3 34	FCB	\$34	
5EC4 43	FCC	'CMP'	



5EC7 34	FCB	\$34
5EC8 53	FCC	'SBC'
5ECB 34	FCB	\$34
5ECC 2A	FCC	'***'
5ECF 00	FCB	00
5ED0 41	FCC	'AND'
5ED3 34	FCB	\$34
5ED4 42	FCC	'BIT'
5ED7 34	FCB	\$34
5ED8 4C	FCC	'LDA'
5EDB 34	FCB	\$34
5EDC 53	FCC	'STA'
5EDF 35	FCB	\$35
5EE0 45	FCC	'EOR'
5EE3 34	FCB	\$34
5EE4 41	FCC	'ADC'
5EE7 34	FCB	\$34
5EE8 4F	FCC	'ORA'
5EEB 34	FCB	\$34
5EEC 41	FCC	'ADD'
5EEF 34	FCB	\$34
5EF0 2A	FCC	'***'
5EF3 00	FCB	00
5EF4 2A	FCC	'***'
5EF7 00	FCB	00
5EF8 4C	FCC	'LDX'
5EFB 35	FCB	\$35
5EFC 53	FCC	'STX'
5EFF 36	FCB	\$36

## \* SECONDARY INFO TABLE

5F00 00	AUXTAB	FCB	\$00	\$00
5F01 50		FCB	\$50	
5F02 00		FCB	\$00	
5F03 00		FCB	\$00	
5F04 00		FCB	\$00	
5F05 00		FCB	\$00	
5F06 50		FCB	\$50	
5F07 50		FCB	\$50	
5F08 50		FCB	\$50	
5F09 50		FCB	\$50	
5F0A 50		FCB	\$50	
5F0B 50		FCB	\$50	
5F0C 50		FCB	\$50	
5F0D 50		FCB	\$50	
5F0E 50		FCB	\$50	
5F0F 50		FCB	\$50	
5F10 50		FCB	\$50	\$10
5F11 50		FCB	\$50	
5F12 00		FCB	\$00	
5F13 00		FCB	\$00	
5F14 00		FCB	\$00	
5F15 00		FCB	\$00	



5F16 50	FCB	\$50
5F17 50	FCB	\$50
5F18 00	FCB	\$00
5F19 50	FCB	\$50
5F1A 00	FCB	\$00
5F1B 50	FCB	\$50
5F1C 00	FCB	\$00
5F1D 00	FCB	\$00
5F1E 00	FCB	\$00
5F1F 00	FCB	\$00
5F20 63	FCB	\$63
5F21 00	FCB	\$00
5F22 63	FCB	\$63
5F23 63	FCB	\$63
5F24 63	FCB	\$63
5F25 63	FCB	\$63
5F26 63	FCB	\$63
5F27 63	FCB	\$63
5F28 63	FCB	\$63
5F29 63	FCB	\$63
5F2A 63	FCB	\$63
5F2B 63	FCB	\$63
5F2C 63	FCB	\$63
5F2D 63	FCB	\$63
5F2E 63	FCB	\$63
5F2F 63	FCB	\$63
5F30 50	FCB	\$50
5F31 50	FCB	\$50
5F32 51	FCB	\$51
5F33 52	FCB	\$52
5F34 50	FCB	\$50
5F35 50	FCB	\$50
5F36 D1	FCB	\$D1
5F37 D2	FCB	\$D2
5F38 00	FCB	\$00
5F39 57	FCB	\$57
5F3A 00	FCB	\$00
5F3B 58	FCB	\$58
5F3C 00	FCB	\$00
5F3D 00	FCB	\$00
5F3E 5A	FCB	\$5A
5F3F 59	FCB	\$59
5F40 51	FCB	\$51
5F41 00	FCB	\$00
5F42 00	FCB	\$00
5F43 51	FCB	\$51
5F44 51	FCB	\$51
5F45 00	FCB	\$00
5F46 51	FCB	\$51
5F47 51	FCB	\$51
5F48 51	FCB	\$51
5F49 51	FCB	\$51
5F4A 51	FCB	\$51
5F4B 00	FCB	\$00

\$20

\$30

\$40



5F4C 51	FCB	\$51	004	001	00 0010
5F4D 51	FCB	\$51	004	001	00 0010
5F4E 00	FCB	\$00	004	000	00 0000
5F4F 51	FCB	\$51	004	001	00 0010
5F50 52	FCB	\$52	004	002	00 0010
5F51 00	FCB	\$00	004	000	00 0000
5F52 00	FCB	\$00	004	000	00 0000
5F53 52	FCB	\$52	004	002	00 0010
5F54 52	FCB	\$52	004	002	00 0010
5F55 00	FCB	\$00	004	000	00 0000
5F56 52	FCB	\$52	004	002	00 0010
5F57 52	FCB	\$52	004	002	00 0010
5F58 52	FCB	\$52	004	002	00 0010
5F59 52	FCB	\$52	004	002	00 0010
5F5A 52	FCB	\$52	004	002	00 0010
5F5B 00	FCB	\$00	004	000	00 0000
5F5C 52	FCB	\$52	004	002	00 0010
5F5D 52	FCB	\$52	004	002	00 0010
5F5E 00	FCB	\$00	004	000	00 0000
5F5F 52	FCB	\$52	004	002	00 0010
5F60 B0	FCB	\$B0	004	000	00 0000
5F61 00	FCB	\$00	004	000	00 0000
5F62 00	FCB	\$00	004	000	00 0000
5F63 B0	FCB	\$B0	004	000	00 0000
5F64 B0	FCB	\$B0	004	000	00 0000
5F65 00	FCB	\$00	004	000	00 0000
5F66 B0	FCB	\$B0	004	000	00 0000
5F67 B0	FCB	\$B0	004	000	00 0000
5F68 B0	FCB	\$B0	004	000	00 0000
5F69 B0	FCB	\$B0	004	000	00 0000
5F6A B0	FCB	\$B0	004	000	00 0000
5F6B 00	FCB	\$00	004	000	00 0000
5F6C B0	FCB	\$B0	004	000	00 0000
5F6D 30	FCB	\$30	004	000	00 0000
5F6E 35	FCB	\$35	004	000	00 0000
5F6F B0	FCB	\$B0	004	000	00 0000
5F70 C0	FCB	\$C0	004	000	00 0000
5F71 00	FCB	\$00	004	000	00 0000
5F72 00	FCB	\$00	004	000	00 0000
5F73 C0	FCB	\$C0	004	000	00 0000
5F74 C0	FCB	\$C0	004	000	00 0000
5F75 00	FCB	\$00	004	000	00 0000
5F76 C0	FCB	\$C0	004	000	00 0000
5F77 C0	FCB	\$C0	004	000	00 0000
5F78 C0	FCB	\$C0	004	000	00 0000
5F79 C0	FCB	\$C0	004	000	00 0000
5F7A C0	FCB	\$C0	004	000	00 0000
5F7B 00	FCB	\$00	004	000	00 0000
5F7C C0	FCB	\$C0	004	000	00 0000
5F7D 40	FCB	\$40	004	000	00 0000
5F7E 45	FCB	\$45	004	000	00 0000
5F7F C0	FCB	\$C0	004	000	00 0000
5F80 11	FCB	\$11	004	000	00 0000
5F81 11	FCB	\$11	004	000	00 0000



5F82 11	FCB	\$11
5F83 00	FCB	\$00
5F84 11	FCB	\$11
5F85 11	FCB	\$11
5F86 11	FCB	\$11
5F87 00	FCB	\$00
5F88 11	FCB	\$11
5F89 11	FCB	\$11
5F8A 11	FCB	\$11
5F8B 11	FCB	\$11
5F8C 10	FCB	\$10
5F8D 64	FCB	\$64
5F8E 10	FCB	\$10
5F8F 00	FCB	\$00
5F90 21	FCB	\$21
5F91 21	FCB	\$21
5F92 21	FCB	\$21
5F93 00	FCB	\$00
5F94 21	FCB	\$21
5F95 21	FCB	\$21
5F96 21	FCB	\$21
5F97 A1	FCB	\$A1
5F98 21	FCB	\$21
5F99 21	FCB	\$21
5F9A 21	FCB	\$21
5F9B 21	FCB	\$21
5F9C 20	FCB	\$20
5F9D 00	FCB	\$00
5F9E 20	FCB	\$20
5F9F A0	FCB	\$A0
5FA0 31	FCB	\$31
5FA1 31	FCB	\$31
5FA2 31	FCB	\$31
5FA3 00	FCB	\$00
5FA4 31	FCB	\$31
5FA5 31	FCB	\$31
5FA6 31	FCB	\$31
5FA7 B1	FCB	\$B1
5FA8 31	FCB	\$31
5FA9 31	FCB	\$31
5FAA 31	FCB	\$31
5FAB 31	FCB	\$31
5FAC 30	FCB	\$30
5FAD 36	FCB	\$36
5FAE 30	FCB	\$30
5FAF B0	FCB	\$B0
5FB0 41	FCB	\$41
5FB1 41	FCB	\$41
5FB2 41	FCB	\$41
5FB3 00	FCB	\$00
5FB4 41	FCB	\$41
5FB5 41	FCB	\$41
5FB6 41	FCB	\$41
5FB7 C1	FCB	\$C1

\$90

\$A0

\$B0



5FB8 41	FCB	\$41
5FB9 41	FCB	\$41
5FBA 41	FCB	\$41
5FBB 41	FCB	\$41
5FBC 40	FCB	\$40
5FBD 46	FCB	\$46
5FBE 40	FCB	\$40
5FBF C0	FCB	\$C0
5FC0 12	FCB	\$12
5FC1 12	FCB	\$12
5FC2 12	FCB	\$12
5FC3 00	FCB	\$00
5FC4 12	FCB	\$12
5FC5 12	FCB	\$12
5FC6 12	FCB	\$12
5FC7 00	FCB	\$00
5FC8 12	FCB	\$12
5FC9 12	FCB	\$12
5FCA 12	FCB	\$12
5FCB 12	FCB	\$12
5FCC 00	FCB	\$00
5FCD 00	FCB	\$00
5FCE 10	FCB	\$10
5FCF 00	FCB	\$00
5FD0 22	FCB	\$22
5FD1 22	FCB	\$22
5FD2 22	FCB	\$22
5FD3 00	FCB	\$00
5FD4 22	FCB	\$22
5FD5 22	FCB	\$22
5FD6 22	FCB	\$22
5FD7 A2	FCB	\$A2
5FD8 22	FCB	\$22
5FD9 22	FCB	\$22
5FDA 22	FCB	\$22
5FDB 22	FCB	\$22
5FDC 00	FCB	\$00
5FDD 00	FCB	\$00
5FDE 20	FCB	\$20
5FDF A0	FCB	\$A0
5FE0 32	FCB	\$32
5FE1 32	FCB	\$32
5FE2 32	FCB	\$32
5FE3 00	FCB	\$00
5FE4 32	FCB	\$32
5FE5 32	FCB	\$32
5FE6 32	FCB	\$32
5FE7 B2	FCB	\$B2
5FE8 32	FCB	\$32
5FE9 32	FCB	\$32
5FER 32	FCB	\$32
5FEB 32	FCB	\$32
5FEC 00	FCB	\$00
5FED 00	FCB	\$00

\$C0

\$D0

\$E0



5FEE 30	FCB	\$30
5FEF 80	FCB	\$80
5FF0 42	FCB	\$42
5FF1 42	FCB	\$42
5FF2 42	FCB	\$42
5FF3 00	FCB	\$00
5FF4 42	FCB	\$42
5FF5 42	FCB	\$42
5FF6 42	FCB	\$42
5FF7 C2	FCB	\$C2
5FF8 42	FCB	\$42
5FF9 42	FCB	\$42
5FFA 42	FCB	\$42
5FFB 42	FCB	\$42
5FFC 00	FCB	\$00
5FFD 00	FCB	\$00
5FFE 40	FCB	\$40
5FFF C0	FCB	\$C0
3C00	ORG	\$3C00

## \* BREAKPOINT TABLE

3C00	BPTAB	RMB	256
3D00	STRTPC	RMB	512
3F00	ENDPC	EQU	*
3F00	SMTAB	RMB	32
3F20	EXTAB	RMB	32
3F40	WPTAB	RMB	32
3F60	MTAB	RMB	32
3F80	MSGTB	RMB	32
3FA0	USER	EQU	*
	END	COLDS	

NO ERROR(S) DETECTED



## SYMBOL TABLE:

ACIA	410F	ADDBX	442D	ADDBX2	443B	AR	408D	ASML	538A
ASML3	53C1	ASML35	53CB	ASML4	53CE	ASML5	53F1	ASML6	53F5
ASML63	5402	ASML65	5404	ASML7	5407	ASML75	541D	ASML77	5423
ASML8	5428	AST	5944	AUXTAB	5F00	BELL	0007	BEXP	50F1
BEXP2	5122	BEXP3	5125	BEXP4	513E	BOUN35	4DBB	BOUND2	4DA0
BOUND3	4DB8	BOUND4	4DBF	BOUND5	4DC3	BOUND6	4DC7	BOUND7	4DC9
BOUND8	4DCE	BOUNDM	4E04	BOUNDS	4D85	BOUNDW	4DF4	BOUNDX	4DE4
BOUNS	4DD4	BOYST	5AC3	BPAD	40B2	BPCODE	40B0	BPEND	40AE
BPP0S	40AC	BPTAB	3C00	BR	408E	BRK	4FB7	BRK2	4FCA
BRK25	4FE0	BRK3	4FE6	BRK4	4FE9	BRK42	5002	BRK45	500E
BRK5	5011	BRK55	5013	BRK58	5044	BRK6	504B	BRK7	5061
BRK72	5071	BSE	4113	BSP	4111	BST	5948	BTYP	40B1
BUFPNT	4050	BXOP	40DA	BXOP2	40DD	CALC	52BE	CALC5	52E6
CALC6	52FE	CALC7	5304	CALC8	530A	CALC9	5322	CC	408C
CCST	5941	CER	4B9D	CER2	4BD3	CER3	4BD9	CER4	4BDC
CER5	4BE2	CER6	4BE4	CFLG	40A4	CHECK	4B77	CHECK2	4B88
CHECK3	4B92	CHECK4	4B94	CHECK5	4B9A	CHECK6	4B9C	CHKSP	56C3
CHKSP1	56CD	CHKSP2	56E5	CHKSP3	56EB	CHKSP4	56FC	CLASS	4342
CLASS2	435A	CLASS4	435F	CLB	522D	CLB2	5230	CLB4	5246
CLCP	5938	CLH	5249	CLH2	524C	CLM	57A0	CLP	54D5
CLP1	54DD	CLP2	54E0	CLP4	54F8	CLP5	5500	CLP6	5508
CLP7	5510	CLP8	5516	CLRPR	417C	CMOD35	4D7F	CMODE	4D72
CMODE3	4D7C	CMODE4	4D82	CNTR	4067	CNUM	5324	CNUM2	5335
CNUM4	5338	CNUM6	5339	COLD	411B	COLD2	4121	COLDS	4100
COMTBL	57A8	CONTBL	58E3	COUNT	4075	CR	000D	CRLFST	5999
CRSAVE	4052	CSIM	525E	CSIM2	5261	CSIM4	5274	CSIM6	5277
CTST	59B4	DATPNT	4056	DCMP	51FE	DCMP2	5204	DEL	4112
DELAY	408B	DELST	5996	DEPTH	56FD	DIS	4E8E	DIS2	4E9F
DIS4	4EB7	DIS6	4EC0	DIS0	4A43	DIS01	4A4B	DIS02	4A59
DIS03	4A79	DIS035	4A7B	DIS04	4A80	DIS045	4A83	DIS047	4A94
DIS05	4AB1	DIS055	4AB5	DIS06	4AD3	DIS067	4AFA	DIS07	4B04
DIS08	4B0A	DOASM	535F	DOASM2	5367	DOASM4	5384	DOASM6	5387
DOCMND	AD4B	DODOS	5589	DOHB	5083	DOHB1	5088	DOHB2	508B
DOIO	4571	DOIOI	454F	DOIOO	455E	DOIOO2	456B	DOJB	50C5
DOMB	508E	DOMB1	5099	DOMB2	50A9	DOMB4	50BB	DOMON	4589
DOSBUF	AC14	DREG	48BC	DSHST	5AE4	DUMP	551F	DUMP1	5524
DUMP2	552F	DUMP25	5540	DUMP27	5551	DUMP28	555B	DUMP5	556D
DUMP6	5586	EADR	40D0	EAFL	40A2	END	4073	ENDPC	3F00
EQUALS	40B9	EQVAL	4C2A	EQVAL2	4C34	ESC	4114	EXEC	4195
EXEC3	41AF	EXEC6	41C2	EXEND	40C8	EXIT	4547	EXPST	59FD
EXTAB	3F20	FILL	5620	FILL2	562C	FIND	55A3	FIND2	55BC
FIND3	55C8	FIND4	55DC	FIND5	55DF	FIND6	55E2	FIND7	5611
FIND8	561D	FIRST	4063	FLAG	5708	FLAG5	5719	FLAGA	409E
FLAGB	40A0	FMSCLS	B403	FMST	5960	FNDM42	547D	FNDM45	5482
FNDMN	5436	FNDMN1	543C	FNDMN2	5456	FNDMN4	5462	FNDMN5	5487
FNDMN6	549C	FNDMN7	54A7	FNDOP	4885	FNDOP2	48B5	FNDOP3	48B6
FNDOP4	48B7	FNDOP5	48B8	FNDOP6	48BB	FNDRE1	4F98	FNDRE2	4F9F
FNDRE4	4FAE	FNDRE6	4FB5	FNDREG	4F8C	FNDTP	516C	FNDTP2	516F
FNDTP4	517B	FNDTP6	517D	FUNTAB	58EB	GETCHR	4106	GETHE2	43BC
GETHE6	43C9	GETHE8	43CF	GETHEX	43B5	GETRN2	4CF8	GETRNG	4CDD
GO	443F	HEQVAL	4C37	HIST	517F	HIST2	5181	HIST6	5194



HIST7	5197	HMONS	5ACC	ILIN	41C6	ILIN2	41C9	ILIN4	41CC
ILOPST	59C3	ILOPTR	487F	INBU65	4280	INBUF	423C	INBUF2	4242
INBUF3	4257	INBUF4	425C	INBUF5	4264	INBUF6	426B	INCOD	40B8
INDEC	43EE	INDEC2	43F5	INDEX	4054	INPST	5988	INRNG	51EE
INST	4C41	INST2	4C51	INV	4107	IRQC	407D	IRQC2	407F
IRQV	4117	ITRAP	54C3	ITRAP2	54CF	ITRAP4	54D2	ITRF	40AB
ITRST	59D6	IXST	5993	JUMP	4446	LAST	4065	LASTJ	40BE
LF	000A	LINBUF	4000	LKBP	444D	LKBP3	4450	LKBP4	445E
LKBP6	4468	LKNA41	41F8	LKNA42	4208	LKNA45	4210	LKNA95	422F
LKNAM	41D3	LKNAM2	41D6	LKNAM3	41D9	LKNAM4	41E6	LKNAM5	4215
LKNAM6	421B	LKNAM7	421C	LKNAM8	422A	LKNAM9	422B	LSTTRM	4068
MACH	5729	MACH2	5758	MACH4	575B	MACH6	5792	MARKRG	40BC
MAXC	4077	MAXSP	4081	MEND	40CC	MMP1ST	5A15	MMP2ST	5A26
MOD	406F	MODE	409B	MONV	410D	MPR	4B6E	MPST	5A61
MSGTB	3F80	MSPST	5A38	MTAB	3F60	NERST	5A89	NEST	4C53
NESTC	4099	NMIC	4079	NMIC2	407B	NMIY	4119	NOBKF	40A3
NORTS	5AA6	NST	5958	NSTER	4E56	NSTER2	4E59	NSTRP	40A5
NSTST	5A7D	NXTBP	40B4	NXTCH	4361	NXTCH2	4364	NXTCH3	4367
NXTCH4	4379	NXTMSG	40B6	NXTPC	40C0	NXTRB	51CE	NXTRB2	51D6
NXTRB4	51EB	OHALT	446B	OLDPC	4097	OP	40D4	OP1LNG	46D5
OP2LNG	46C2	OP3LNG	46C9	OPCNT	409A	OPHST	5AD8	OPJMP	40D7
OPND	4070	OPPNT	40C2	OPPNT2	40C4	OPTAB	5B00	OUT2SP	494D
OUTADR	4326	OUTDE2	42D8	OUTDE4	42E7	OUTDEC	42D5	OUTDI2	42F6
OUTDI4	4300	OUTDI5	4309	OUTDI6	431F	OUTDI8	4324	OUTDIG	42F3
OUTHEX	4329	OUTHL	4331	OUTHR	4335	OUTHR2	433F	OUTNUM	405A
OUTSP	494F	OUTST	598D	OUTV	410A	PADR	4AA4	PADR2	4AA7
PAST	4C8C	PAST2	4CA2	PAST25	4CA5	PAST3	4CAD	PAST4	4CB4
PAST5	4CCF	PAST6	4CDA	PAUSF	408A	PBP	5205	PBP2	5208
PBP3	5219	PBP4	5221	PBP6	522A	PBPAD	51B6	PBPRT	4E32
PBRA	473F	PBRA4	4757	PBSR	4730	PBXX	4758	PBYTE	4AAC
PC	4093	PCRLF	4294	PCRLF2	42AD	PCST	5954	PDATA1	4287
PDATA2	4293	PERR	4AA9	PIRQ	481D	PIRQ1	482D	PIRQ2	482F
PIRQ4	4834	PJMP	478A	PJMPX	47A0	PJSR	477D	PJSRX	479E
PNMI	483A	PNMI2	4842	POINT	405F	POINT2	4061	PONS	47D4
PONS2	4806	POPTAB	40D2	PRB45	499E	PRB46	49A7	PRB47	49B2
PRB475	49B7	PRB48	49C3	PRB49	49C7	PRB495	49D0	PRB55	49DA
PRB57	49DF	PRB62	49FD	PRB65	4A07	PRB68	4A17	PRB75	4A34
PRBP	4954	PRBP1	496A	PRBP2	4976	PRBP3	4987	PRBP4	498A
PRBP5	49D7	PRBP6	49EE	PRBP7	4A24	PRBP9	4A39	PRCH	4B12
PRCH2	4B2A	PRCH3	4B34	PRCH4	4B40	PRCH5	4B62	PRMPT	5935
PRON	409C	PRONF	5794	PRONF2	579B	PROT	4CF9	PROT2	4CFB
PROT3	4D10	PROT35	4D13	PROT4	4D22	PROT5	4D31	PROT6	4D40
PROT7	4D4D	PROT8	4D57	PRRE22	490D	PRRE25	4910	PRREG	48C7
PRREG2	48D1	PRREG3	4936	PRREG4	4946	PRTB	4E12	PRTB1	4E15
PRTB2	4E2F	PRTI	4851	PRTSTR	5A71	PRVW	4394	PSTAT	527A
PSTAT4	528A	PSTK	533B	PSTK0	5347	PSTK1	534A	PSTK2	534E
PSTRNG	4285	PSUBC	4765	PSUBC2	477C	PSUBR	47B0	PSWI	4810
PSWI2	4819	PSWI4	481C	PUTCHR	4109	PWAI	4849	PWAI2	484B
RELADR	40BA	RGT	5907	RGTAB	58F9	RMVBP	514D	RMVBP1	5152
RMVBP2	515E	RMVBP4	516B	RT	5636	RT2	5642	RT4	5684
RTAD	406A	RTADS	5A9C	RTB	5687	RTB2	56A8	RTB4	56BD
RTB6	56C0	RTBF	4069	RTDAT	406C	RTRN	4E5C	RTRN1	4E80
RTRN2	4E83	RTRN4	4E8B	SDELY	528C	SEIM	4807	SET	4F48
SET2	4F4D	SET4	4F64	SET5	4F81	SET6	4F89	SETCN	50D9



SETCN2	50E9	SETCN4	50EE	SETFL	519A	SETFL2	51AF	SETV	4D5D
SIMCNT	4087	SIMU15	4488	SIMU35	44B8	SIMU45	44D6	SIMU48	44E2
SIMU61	4504	SIMU62	4507	SIMU65	450D	SIMU67	4524	SIMU68	4529
SIMU69	452C	SIMUL	4471	SIMUL0	4476	SIMUL1	4479	SIMUL2	4490
SIMUL3	449E	SIMUL4	44C8	SIMUL5	44EC	SIMUL6	44FF	SIMUL7	453E
SIND	5297	SIND2	52A3	SINDEC	571C	SIRQ	52A8	SKPSP2	4384
SKPSP4	438D	SKPSPC	437E	SMEND	40C6	SMPST	5A68	SMTAB	3F00
SNMI	52B3	SP	4091	SPC	0020	SPST	5950	SSM	4BE6
SSM1	4BFC	SSM15	4C02	SSM2	4C27	SSM3	4C17	SSP	4095
SSTOP	4ECD	SSTOP2	4EDC	STACK	3FFF	STATES	40A7	STOP	454A
STPCNT	4085	STPST	59A9	STRAC	4EC2	STRIT	4D6B	STRTPC	3D00
STST	599F	STXFR	4E40	STXFR2	4E4A	SWIV	4115	SYNER	41CE
SYNST	59F0	TABE	40CE	TBLOV	5A44	TCSIM	5257	TEMP	405B
TEMP1	405C	TRAIL	4C7E	TRC125	4676	TRCF	409D	TRCHK	45E7
TRCHK1	45F9	TRCHK2	45FC	TRCHK4	45FF	TRCHK5	4607	TRCHK6	4615
TRCHK7	4641	TRC011	45D4	TRC012	4657	TRC013	468D	TRC014	469E
TRC015	46AC	TRC045	46F9	TRC047	46FE	TRCOL	458E	TRCOL1	45CC
TRCOL2	46B2	TRCOL3	46D4	TRCOL4	46F2	TRCOL5	4702	TRCRET	471A
TRCSP1	466D	TRCSP2	4673	TRNEST	4E3B	TRST	59E1	TSP	4083
TSTEQ	4C5D	TSTHE1	43DA	TSTHE2	43E6	TSTHE3	43EA	TSTHE4	43EC
TSTHEX	43D4	TSTON	4C69	TSTTRM	4C63	TYPTB	5924	USER	3FA0
VALUE	405D	VFLG	40A1	WAITF	4089	WAITR	42B6	WAITR1	42BC
WAITR2	42D2	WARM	4195	WARMS	410C	WARMST	4103	WHATST	593B
WPEND	40CA	WPRST	5A53	WPST	5A09	WPTAB	3F40	WRKHx	43A0
WRKHx4	43A3	XFRST	5AB6	XFRT	40A6	XFRTRP	4E51	XMEM	4EDE
XMEM2	4EEA	XMEM3	4EED	XMEM35	4EF3	XMEM37	4F19	XMEM38	4F2A
XMEM4	4F3A	XMEM5	4F3D	XMIRO	5975	XMIT	596B	XMM	5964
XNMNI	597B	XMT	5967	XMXT	5970	XOFST	5984	XONST	5981
XPST	5A59	XR	408F	XSAVE	4058	XST	594C	XVST	595C